Contents

1 GAMS Documentation Center 1
  1.1 Model Libraries .................................................. 1
  1.2 Further Help .................................................... 2

2 Preface 3
  2.1 Timeline for documentation released with GAMS software ............... 4

3 Release Notes 7
  3.1 Release Types .................................................. 7
  3.2 Release History .................................................. 7
  3.3 40 Distribution .................................................. 12
    3.3.1 40.1.0 Major release (August 01, 2022) .............. 12
  3.4 39 Distribution .................................................. 20
    3.4.1 39.1.0 Major release (May 03, 2022) .............. 20
    3.4.2 39.1.1 Maintenance release (May 18, 2022) ...... 31
    3.4.3 39.2.0 Minor release (June 02, 2022) ............ 32
    3.4.4 39.2.1 Maintenance release (June 21, 2022) ...... 33
    3.4.5 39.3.0 Minor release (July 07, 2022) ............ 34
  3.5 38 Distribution .................................................. 35
    3.5.1 38.1.0 Major release (January 31, 2022) .......... 35
    3.5.2 38.2.0 Minor release (February 17, 2022) ....... 47
    3.5.3 38.2.1 Maintenance release (February 19, 2022) .. 49
    3.5.4 38.3.0 Minor release (April 05, 2022) ......... 50
  3.6 37 Distribution .................................................. 53
    3.6.1 37.1.0 Major release (November 11, 2021) .......... 53
  3.7 36 Distribution .................................................. 60
    3.7.1 36.1.0 Major release (August 02, 2021) .......... 60
    3.7.2 36.2.0 Minor release (September 03, 2021) ...... 71
  3.8 35 Distribution .................................................. 73
    3.8.1 35.1.0 Major release (April 29, 2021) ........... 73
    3.8.2 35.2.0 Minor release (June 02, 2021) ........... 90
  3.9 34 Distribution .................................................. 92
    3.9.1 34.1.0 Major release (January 29, 2021) .......... 92
    3.9.2 34.2.0 Minor release (February 05, 2021) ...... 104
    3.9.3 34.3.0 Minor release (February 25, 2021) ...... 105
  3.10 33 Distribution .................................................. 107
    3.10.1 33.1.0 Major release (November 01, 2020) ...... 107
    3.10.2 33.2.0 Minor release (December 01, 2020) ...... 116
  3.11 32 Distribution .................................................. 117
    3.11.1 32.1.0 Major release (July 31, 2020) ........... 117
    3.11.2 32.2.0 Minor release (August 26, 2020) ...... 124
  3.12 31 Distribution .................................................. 126
    3.12.1 31.1.0 Major release (May 01, 2020) ........... 126
    3.12.2 31.1.1 Maintenance release (May 16, 2020) ..... 138
    3.12.3 31.2.0 Minor release (June 19, 2020) ........ 139
  3.13 30 Distribution .................................................. 140
4 User's Guide

4.1 Installation and Licensing .................................................. 295
4.2 Tutorials and Examples ....................................................... 295
4.3 GAMS Language and Environment ......................................... 296
4.4 Glossary ............................................................................ 297
4.5 Supported Platforms ........................................................... 298
4.6 Installation Notes for Mac OS X ............................................. 299
  4.6.1 Installation using the PKG installer (GAMS40.1.0.pkg) .......... 299
  4.6.2 Uninstall PKG installation ............................................... 301
  4.6.3 Installation using the self-extracting archive (osx_x64_64.sfx.exe) .. 301
4.7 Installation Notes for Unix ..................................................... 305
  4.7.1 Installation .................................................................. 305
  4.7.2 Access to GAMS ............................................................ 306
4.8 Installation Notes for Windows ............................................... 307
  4.8.1 Installation .................................................................. 307
  4.8.2 Visual C++ Redistributable Dependency ............................ 309
  4.8.3 Command Line Use of GAMS .......................................... 309
  4.8.4 Warning from Microsoft SmartScreen Filter ....................... 309
4.9 Standard Locations ............................................................... 310
  4.9.1 Standard Locations on Mac OS X ...................................... 310
  4.9.2 Standard Locations on Unix ............................................. 311
  4.9.3 Standard Locations on Windows ....................................... 312
4.10 Licensing ........................................................................... 313
  4.10.1 General Information ..................................................... 313
  4.10.2 GAMS Community License ............................................ 313
  4.10.3 GAMS Licenses ............................................................ 313
  4.10.4 Installing or updating a license file .................................. 314
  4.10.5 License Problems ......................................................... 314
  4.10.6 Warnings .................................................................. 315
  4.10.7 The GAMS/BASE Module ............................................. 316
  4.10.8 Additional Limits for the Demo and Community License ...... 317
4.11 A GAMS Tutorial by Richard E. Rosenthal ............................... 318
  4.11.1 Introduction .................................................................. 318
  4.11.2 Structure of a GAMS Model ............................................ 320
  4.11.3 Sets ............................................................................ 322
  4.11.4 Data ........................................................................... 323
  4.11.5 Variables ..................................................................... 325
  4.11.6 Equations .................................................................... 326
  4.11.7 Objective Function ...................................................... 329
  4.11.8 Model and Solve Statements .......................................... 329
  4.11.9 Display Statements ...................................................... 330
  4.11.10 The .lo, .l, .up, .m Database ......................................... 330
<table>
<thead>
<tr>
<th>4.20.3 Using STUDIO after Installation</th>
<th>488</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.20.4 The Welcome and Explorer Windows</td>
<td>501</td>
</tr>
<tr>
<td>4.20.5 Fixing Compilation Errors</td>
<td>501</td>
</tr>
<tr>
<td>4.20.6 Ways to find and/or replace text strings</td>
<td>503</td>
</tr>
<tr>
<td>4.20.7 Matching Parentheses</td>
<td>504</td>
</tr>
<tr>
<td>4.20.8 Moving Blocks</td>
<td>505</td>
</tr>
<tr>
<td>4.20.9 Syntax Coloring</td>
<td>506</td>
</tr>
<tr>
<td>4.20.10 Showing where a symbol appears</td>
<td>507</td>
</tr>
<tr>
<td>4.20.11 Accessing Documentation Via the Help</td>
<td>507</td>
</tr>
<tr>
<td>4.20.12 Making GDX files</td>
<td>509</td>
</tr>
<tr>
<td>4.20.13 Examining GDX files</td>
<td>510</td>
</tr>
<tr>
<td>4.20.14 A difficulty you will have</td>
<td>512</td>
</tr>
<tr>
<td>4.20.15 Command Line Parameters</td>
<td>512</td>
</tr>
<tr>
<td>4.20.16 Solver Option Files</td>
<td>514</td>
</tr>
<tr>
<td>4.20.17 Using Libraries</td>
<td>514</td>
</tr>
<tr>
<td>4.20.18 Using reference files - Listing and Unraveling Data items</td>
<td>515</td>
</tr>
<tr>
<td>4.20.19 Editing Solver Option Files</td>
<td>521</td>
</tr>
<tr>
<td>4.20.20 When is it Not Worth Using?</td>
<td>522</td>
</tr>
<tr>
<td>4.20.21 What does it not do?</td>
<td>522</td>
</tr>
<tr>
<td>4.20.22 What does it not do so well?</td>
<td>522</td>
</tr>
<tr>
<td>4.21 Introduction</td>
<td>523</td>
</tr>
<tr>
<td>4.21.1 Summary</td>
<td>523</td>
</tr>
<tr>
<td>4.21.2 The Origins of GAMS</td>
<td>523</td>
</tr>
<tr>
<td>4.21.3 Background and Motivation</td>
<td>523</td>
</tr>
<tr>
<td>4.21.4 Design Goals and Changing Focus</td>
<td>524</td>
</tr>
<tr>
<td>4.22 GAMS Programs</td>
<td>525</td>
</tr>
<tr>
<td>4.22.1 Introduction</td>
<td>525</td>
</tr>
<tr>
<td>4.22.2 The Structure of GAMS Programs</td>
<td>526</td>
</tr>
<tr>
<td>4.22.3 Data Types and Definitions</td>
<td>530</td>
</tr>
<tr>
<td>4.22.4 Language Items</td>
<td>531</td>
</tr>
<tr>
<td>4.22.5 Summary</td>
<td>541</td>
</tr>
<tr>
<td>4.23 Set Definition</td>
<td>541</td>
</tr>
<tr>
<td>4.23.1 Introduction</td>
<td>541</td>
</tr>
<tr>
<td>4.23.2 Simple Sets</td>
<td>541</td>
</tr>
<tr>
<td>4.23.3 The Alias Statement: Multiple Names for a Set</td>
<td>544</td>
</tr>
<tr>
<td>4.23.4 Subsets</td>
<td>545</td>
</tr>
<tr>
<td>4.23.5 Multi-Dimensional Sets</td>
<td>546</td>
</tr>
<tr>
<td>4.23.6 Singleton Sets</td>
<td>550</td>
</tr>
<tr>
<td>4.23.7 The Universal Set: * as Set Identifier</td>
<td>551</td>
</tr>
<tr>
<td>4.23.8 Set and Set Element Referencing</td>
<td>552</td>
</tr>
<tr>
<td>4.23.9 Set Attributes</td>
<td>552</td>
</tr>
<tr>
<td>4.23.10 Finding Sets from Data</td>
<td>554</td>
</tr>
<tr>
<td>4.23.11 Domain Checking</td>
<td>555</td>
</tr>
<tr>
<td>4.23.12 Implicit Set Definition (or: Domain Defining Symbol Declarations)</td>
<td>556</td>
</tr>
<tr>
<td>4.23.13 Summary</td>
<td>558</td>
</tr>
<tr>
<td>4.24 Dynamic Sets</td>
<td>558</td>
</tr>
<tr>
<td>4.24.1 Introduction</td>
<td>558</td>
</tr>
<tr>
<td>4.24.2 Assigning Membership to Dynamic Sets</td>
<td>559</td>
</tr>
<tr>
<td>4.24.3 Set Operations</td>
<td>562</td>
</tr>
<tr>
<td>4.24.4 Using Dollar Controls with Dynamic Sets</td>
<td>563</td>
</tr>
<tr>
<td>4.25 Sets as Sequences: Ordered Sets</td>
<td>568</td>
</tr>
<tr>
<td>4.25.1 Introduction</td>
<td>568</td>
</tr>
<tr>
<td>4.25.2 Ordered and Unordered Sets</td>
<td>568</td>
</tr>
<tr>
<td>4.25.3 Sorting a Set</td>
<td>569</td>
</tr>
<tr>
<td>4.25.4 Ord and Card Operators</td>
<td>570</td>
</tr>
<tr>
<td>4.25.5 Lag and Lead Operators</td>
<td>571</td>
</tr>
<tr>
<td>4.25.6 Summary</td>
<td>577</td>
</tr>
<tr>
<td>4.26 Data Manipulations with Parameters</td>
<td>577</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>4.26.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.26.2</td>
<td>The Assignment Statement</td>
</tr>
<tr>
<td>4.26.3</td>
<td>Expressions</td>
</tr>
<tr>
<td>4.26.4</td>
<td>Functions</td>
</tr>
<tr>
<td>4.26.5</td>
<td>Extended Range Arithmetic and Error Handling</td>
</tr>
<tr>
<td>4.26.6</td>
<td>Predefined Symbols</td>
</tr>
<tr>
<td>4.26.7</td>
<td>Summary</td>
</tr>
<tr>
<td>4.27</td>
<td>Data Entry: Parameters, Scalars and Tables</td>
</tr>
<tr>
<td>4.27.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.27.2</td>
<td>Scalars</td>
</tr>
<tr>
<td>4.27.3</td>
<td>Parameters</td>
</tr>
<tr>
<td>4.27.4</td>
<td>Tables</td>
</tr>
<tr>
<td>4.27.5</td>
<td>Constant Evaluation</td>
</tr>
<tr>
<td>4.27.6</td>
<td>Data Entry by Assignment</td>
</tr>
<tr>
<td>4.27.7</td>
<td>Acronyms</td>
</tr>
<tr>
<td>4.27.8</td>
<td>Summary</td>
</tr>
<tr>
<td>4.28</td>
<td>Variables</td>
</tr>
<tr>
<td>4.28.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.28.2</td>
<td>Variable Declarations</td>
</tr>
<tr>
<td>4.28.3</td>
<td>Variable Attributes</td>
</tr>
<tr>
<td>4.28.4</td>
<td>Variables in Display and Assignment Statements</td>
</tr>
<tr>
<td>4.28.5</td>
<td>Summary</td>
</tr>
<tr>
<td>4.29</td>
<td>Equations</td>
</tr>
<tr>
<td>4.29.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.29.2</td>
<td>Declaring Equations</td>
</tr>
<tr>
<td>4.29.3</td>
<td>Defining Equations</td>
</tr>
<tr>
<td>4.29.4</td>
<td>Expressions in Equation Definitions</td>
</tr>
<tr>
<td>4.29.5</td>
<td>Equation Attributes</td>
</tr>
<tr>
<td>4.29.6</td>
<td>Summary and Quick Reference</td>
</tr>
<tr>
<td>4.30</td>
<td>Model and Solve Statements</td>
</tr>
<tr>
<td>4.30.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.30.2</td>
<td>The Model Statement</td>
</tr>
<tr>
<td>4.30.3</td>
<td>The Solve Statement</td>
</tr>
<tr>
<td>4.30.4</td>
<td>Programs with Several Solve Statements</td>
</tr>
<tr>
<td>4.30.5</td>
<td>Choosing a Solver</td>
</tr>
<tr>
<td>4.30.6</td>
<td>Making New Solvers Available with GAMS</td>
</tr>
<tr>
<td>4.31</td>
<td>Conditional Expressions, Assignments and Equations</td>
</tr>
<tr>
<td>4.31.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.31.2</td>
<td>The Dollar Condition</td>
</tr>
<tr>
<td>4.31.3</td>
<td>Logical Conditions</td>
</tr>
<tr>
<td>4.31.4</td>
<td>Conditional Assignments</td>
</tr>
<tr>
<td>4.31.5</td>
<td>Conditional Indexed Operations</td>
</tr>
<tr>
<td>4.31.6</td>
<td>Conditional Equations</td>
</tr>
<tr>
<td>4.31.7</td>
<td>Filtering Sets</td>
</tr>
<tr>
<td>4.32</td>
<td>The Display Statement</td>
</tr>
<tr>
<td>4.32.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.32.2</td>
<td>The Syntax</td>
</tr>
<tr>
<td>4.32.3</td>
<td>Displaying Multi-Dimensional Identifiers: Label Order</td>
</tr>
<tr>
<td>4.32.4</td>
<td>Display Controls</td>
</tr>
<tr>
<td>4.32.5</td>
<td>Conditional Displays</td>
</tr>
<tr>
<td>4.33</td>
<td>Programming Flow Control Features</td>
</tr>
<tr>
<td>4.33.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.33.2</td>
<td>The If Statement</td>
</tr>
<tr>
<td>4.33.3</td>
<td>The Loop Statement</td>
</tr>
<tr>
<td>4.33.4</td>
<td>The While Statement</td>
</tr>
<tr>
<td>4.33.5</td>
<td>The For Statement</td>
</tr>
<tr>
<td>4.33.6</td>
<td>The Repeat Statement</td>
</tr>
<tr>
<td>4.33.7</td>
<td>The Break Statement</td>
</tr>
</tbody>
</table>
### Contents

1. **4.34 The Option Statement**
   - 4.34.1 Introduction ......................................................... 695
   - 4.34.2 List of Options .................................................... 697

2. **4.35 System Attributes**
   - 4.35.1 Introduction ......................................................... 709
   - 4.35.2 System Suffixes ..................................................... 709
   - 4.35.3 System Data ........................................................ 713
   - 4.35.4 Access to Hidden Functions ...................................... 715

3. **4.36 The Grid and Multi-Threading Solve Facility**
   - 4.36.1 Introduction ......................................................... 715
   - 4.36.2 The Grid Facility: Basic Concepts ................................. 716
   - 4.36.3 The Grid Facility: A First Example ................................ 716
   - 4.36.4 Advanced Use of Grid Features .................................... 719
   - 4.36.5 Summary of Grid Features ......................................... 721
   - 4.36.6 The Grid Facility: Architecture and Customization .......... 724
   - 4.36.7 Multi-Threading ..................................................... 726

4. **4.37 Special Features for Mathematical Programs**
   - 4.37.1 Introduction ......................................................... 727
   - 4.37.2 Special Mixed Integer Programming (MIP) Features ............. 727
   - 4.37.3 Model Scaling - The Scale Option ................................ 735
   - 4.37.4 Conic Programming in GAMS ....................................... 738
   - 4.37.5 Indicator Constraints ............................................... 742

5. **4.38 GAMS Output**
   - 4.38.1 Introduction ......................................................... 746
   - 4.38.2 An Illustrative Model .............................................. 746
   - 4.38.3 Compilation Output ................................................ 747
   - 4.38.4 Execution Output .................................................. 754
   - 4.38.5 Model Generation Output .......................................... 754
   - 4.38.6 The Solution Report ............................................... 758
   - 4.38.7 Post-Solution Output .............................................. 764
   - 4.38.8 Error Reporting .................................................... 765
   - 4.38.9 Customizing the Output File ..................................... 769

6. **4.39 GAMS Log**
   - 4.39.1 Introduction ......................................................... 772
   - 4.39.2 Header ............................................................. 772
   - 4.39.3 Compilation Log .................................................... 773
   - 4.39.4 Execution Output .................................................. 773

7. **4.40 The GAMS Call and Command Line Parameters**
   - 4.40.1 The Generic GAMS Call ............................................ 775
   - 4.40.2 Double Dash Parameters, Compile-Time Variables and Environment Variables ......................................................... 784
   - 4.40.3 Compile-Time Constants .......................................... 789
   - 4.40.4 GAMS Compile Time and Execution Time Phase ................. 791
   - 4.40.5 List of Command Line Parameters ................................ 792
   - 4.40.6 Detailed Descriptions of All Options ........................... 797
   - 4.40.7 Executing an External Program ................................ 870

8. **4.41 Dollar Control Options**
   - 4.41.1 Introduction ......................................................... 871
   - 4.41.2 List of Dollar Control Options ................................... 872
   - 4.41.3 Detailed Description of Dollar Control Options ................ 878
   - 4.41.4 Conditional Compilation ........................................... 969
   - 4.41.5 Macros in GAMS ..................................................... 977
   - 4.41.6 Compressing and Decompressing Files ............................ 982
   - 4.41.7 Encrypting Files .................................................... 984

9. **4.42 The Put Writing Facility**
   - 4.42.1 Introduction ......................................................... 988
   - 4.42.2 The Syntax ........................................................ 988
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.42.3 A First Example</td>
<td>991</td>
</tr>
<tr>
<td>4.42.4 Put Files</td>
<td>992</td>
</tr>
<tr>
<td>4.42.5 Put File Pages</td>
<td>1002</td>
</tr>
<tr>
<td>4.42.6 Output Items</td>
<td>1008</td>
</tr>
<tr>
<td>4.42.7 The Put Utility Statement</td>
<td>1021</td>
</tr>
<tr>
<td>4.42.8 Conditional Put Statements</td>
<td>1029</td>
</tr>
<tr>
<td>4.42.9 Errors Associated with Put Statements</td>
<td>1030</td>
</tr>
<tr>
<td>4.42.10 Creating a Report for the Model MEXSS</td>
<td>1031</td>
</tr>
<tr>
<td>4.43 Solver Usage</td>
<td>1034</td>
</tr>
<tr>
<td>4.43.1 Controlling a Solver via GAMS Options</td>
<td>1035</td>
</tr>
<tr>
<td>4.43.2 The Solver Options File</td>
<td>1035</td>
</tr>
<tr>
<td>4.43.3 Starting Point and Initial Basis</td>
<td>1037</td>
</tr>
<tr>
<td>4.43.4 Trace Features</td>
<td>1039</td>
</tr>
<tr>
<td>4.43.5 Branch-and-Cut-and-Heuristic Facility (BCH)</td>
<td>1046</td>
</tr>
<tr>
<td>4.43.6 Choosing an appropriate Solver</td>
<td>1051</td>
</tr>
<tr>
<td>4.44 The Save and Restart Feature</td>
<td>1053</td>
</tr>
<tr>
<td>4.44.1 Basic Usage</td>
<td>1053</td>
</tr>
<tr>
<td>4.44.2 Use Cases</td>
<td>1056</td>
</tr>
<tr>
<td>4.44.3 Secure Work Files</td>
<td>1061</td>
</tr>
<tr>
<td>4.44.4 Obfuscated Work Files</td>
<td>1067</td>
</tr>
<tr>
<td>4.45 Embedded Code Facility</td>
<td>1069</td>
</tr>
<tr>
<td>4.45.1 Motivation</td>
<td>1069</td>
</tr>
<tr>
<td>4.45.2 Concept</td>
<td>1069</td>
</tr>
<tr>
<td>4.45.3 Simple Example</td>
<td>1070</td>
</tr>
<tr>
<td>4.45.4 Syntax</td>
<td>1074</td>
</tr>
<tr>
<td>4.45.5 Python</td>
<td>1076</td>
</tr>
<tr>
<td>4.45.6 Connect</td>
<td>1091</td>
</tr>
<tr>
<td>4.46 GAMS Connect</td>
<td>1093</td>
</tr>
<tr>
<td>4.46.1 Concept</td>
<td>1093</td>
</tr>
<tr>
<td>4.46.2 Usage</td>
<td>1093</td>
</tr>
<tr>
<td>4.46.3 Connect Agents Summary</td>
<td>1095</td>
</tr>
<tr>
<td>4.46.4 Getting Started</td>
<td>1096</td>
</tr>
<tr>
<td>4.46.5 Connect Agents</td>
<td>1104</td>
</tr>
<tr>
<td>4.46.6 Text Substitutions in YAML File</td>
<td>1131</td>
</tr>
<tr>
<td>4.46.7 Use Connect Agents in Custom Python Code</td>
<td>1131</td>
</tr>
<tr>
<td>4.46.8 Command Line Utility gamsconnect</td>
<td>1133</td>
</tr>
<tr>
<td>4.47 Extrinsic Functions</td>
<td>1133</td>
</tr>
<tr>
<td>4.47.1 Introduction</td>
<td>1133</td>
</tr>
<tr>
<td>4.47.2 Using Function Libraries</td>
<td>1133</td>
</tr>
<tr>
<td>4.47.3 Libraries that are included in the GAMS Distribution</td>
<td>1134</td>
</tr>
<tr>
<td>4.47.4 Build Your Own Library</td>
<td>1145</td>
</tr>
<tr>
<td>4.47.5 Extrinsic Functions vs. External Equations</td>
<td>1148</td>
</tr>
<tr>
<td>4.48 External Equations</td>
<td>1148</td>
</tr>
<tr>
<td>4.48.1 Examples in the GAMS Test Library</td>
<td>1149</td>
</tr>
<tr>
<td>4.48.2 Model Interface</td>
<td>1150</td>
</tr>
<tr>
<td>4.48.3 Programming Interface</td>
<td>1152</td>
</tr>
<tr>
<td>4.48.4 Implementation</td>
<td>1155</td>
</tr>
<tr>
<td>4.49 GAMS Return Codes</td>
<td>1160</td>
</tr>
<tr>
<td>4.49.1 List of the Error/Return Codes</td>
<td>1161</td>
</tr>
<tr>
<td>4.50 GAMS Data eXchange (GDX)</td>
<td>1162</td>
</tr>
<tr>
<td>4.50.1 Reading a GDX file</td>
<td>1163</td>
</tr>
<tr>
<td>4.50.2 Writing a GDX file</td>
<td>1176</td>
</tr>
<tr>
<td>4.50.3 General notes on GDX files</td>
<td>1183</td>
</tr>
<tr>
<td>4.50.4 GAMS Data eXchange Tools</td>
<td>1183</td>
</tr>
<tr>
<td>4.51 Extended Mathematical Programming (EMP)</td>
<td>1184</td>
</tr>
<tr>
<td>4.51.1 EMP Annotations: the EMP Info File</td>
<td>1185</td>
</tr>
<tr>
<td>4.51.2 Soft Constraints</td>
<td>1186</td>
</tr>
<tr>
<td>4.51.3 Variational Inequalities (VI)</td>
<td>1188</td>
</tr>
</tbody>
</table>
5 Solver Manuals

5.1 Model Types ................................................. 1384
5.2 Supported Platforms ......................................... 1386
5.3 AlphaECP .................................................. 1387
  5.3.1 Introduction ......................................... 1387
  5.3.2 GAMS/AlphaECP Output ................................ 1388
  5.3.3 Notes about Options .................................. 1392
  5.3.4 Summary of AlphaECP Options ...................... 1392
  5.3.5 Detailed Descriptions of AlphaECP Options .... 1394
  5.3.6 FAQ .................................................. 1401
5.4 ANTIGONE ................................................. 1402
  5.4.1 Introduction ......................................... 1402
  5.4.2 GAMS/ANTIGONE Output ......................... 1403
  5.4.3 Summary of ANTIGONE Options ................. 1405
  5.4.4 Detailed Descriptions of ANTIGONE Options .. 1407
  5.4.5 ANTIGONE Algorithmic Features ............ 1412
5.5 BARON .................................................. 1414
  5.5.1 Introduction ......................................... 1414
  5.5.2 Model requirements ................................ 1415
  5.5.3 BARON output ...................................... 1416
  5.5.4 Some BARON features ........................... 1419
  5.5.5 The BARON options ....................... 1422
5.6 CBC ................................................... 1429
  5.6.1 Usage .............................................. 1429
  5.6.2 List of Options ..................................... 1430
  5.6.3 Detailed Options Description ............... 1434
5.7 CONOPT ................................................. 1460
  5.7.1 Introduction ......................................... 1460
  5.7.2 Iteration Output .................................... 1461
  5.7.3 GAMS/CONOPT Termination Messages .. 1463
  5.7.4 Function Evaluation Errors .................... 1467
  5.7.5 The CONOPT Options File .......................... 1468
  5.7.6 Hints on Good Model Formulation ............ 1468
  5.7.7 NLP and DNLP Models .......................... 1475
  5.7.8 APPENDIX A: Algorithmic Information .... 1479
  5.7.9 APPENDIX B - Options ....................... 1502
  5.7.10 APPENDIX C: References ..................... 1521
5.8 CONOPT4 ................................................. 1521
  5.8.1 Introduction ......................................... 1521
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8.2 When should you use CONOPT4?</td>
<td>1521</td>
</tr>
<tr>
<td>5.8.3 Memory Management</td>
<td>1522</td>
</tr>
<tr>
<td>5.8.4 Revised Preprocessor</td>
<td>1522</td>
</tr>
<tr>
<td>5.8.5 Phase 0 - Finding an Initial Feasible Solution</td>
<td>1525</td>
</tr>
<tr>
<td>5.8.6 Transition between SLP and SQP</td>
<td>1526</td>
</tr>
<tr>
<td>5.8.7 Bad Iterations</td>
<td>1526</td>
</tr>
<tr>
<td>5.8.8 Saddle Points and Directions of Negative Curvature</td>
<td>1526</td>
</tr>
<tr>
<td>5.8.9 Use of Alternative Sub-Models</td>
<td>1527</td>
</tr>
<tr>
<td>5.8.10 Multiple Threads</td>
<td>1531</td>
</tr>
<tr>
<td>5.8.11 Solving CNS Models</td>
<td>1531</td>
</tr>
<tr>
<td>5.8.12 APPENDIX A - Options</td>
<td>1538</td>
</tr>
<tr>
<td>5.9 CONVERT</td>
<td>1555</td>
</tr>
<tr>
<td>5.9.1 Introduction</td>
<td>1555</td>
</tr>
<tr>
<td>5.9.2 How to use CONVERT</td>
<td>1555</td>
</tr>
<tr>
<td>5.9.3 The GAMS Scalar Format</td>
<td>1556</td>
</tr>
<tr>
<td>5.9.4 The OSiL Format</td>
<td>1559</td>
</tr>
<tr>
<td>5.9.5 User-Specified Options</td>
<td>1559</td>
</tr>
<tr>
<td>5.10 COPT</td>
<td>1561</td>
</tr>
<tr>
<td>5.10.1 Usage</td>
<td>1561</td>
</tr>
<tr>
<td>5.10.2 List of COPT Options</td>
<td>1565</td>
</tr>
<tr>
<td>5.11 CPLEX</td>
<td>1569</td>
</tr>
<tr>
<td>5.11.1 Introduction</td>
<td>1569</td>
</tr>
<tr>
<td>5.11.2 How to Run a Model with Cplex</td>
<td>1569</td>
</tr>
<tr>
<td>5.11.3 Overview of Cplex</td>
<td>1570</td>
</tr>
<tr>
<td>5.11.4 GAMS Options</td>
<td>1576</td>
</tr>
<tr>
<td>5.11.5 Summary of CPLEX Options</td>
<td>1578</td>
</tr>
<tr>
<td>5.11.6 Special Notes</td>
<td>1588</td>
</tr>
<tr>
<td>5.11.7 GAMS/Cplex Log File</td>
<td>1593</td>
</tr>
<tr>
<td>5.11.8 Detailed Descriptions of CPLEX Options</td>
<td>1597</td>
</tr>
<tr>
<td>5.11.9 Setting up a GAMS/Cplex-Link license</td>
<td>1653</td>
</tr>
<tr>
<td>5.12 Deterministic Equivalent (DE)</td>
<td>1653</td>
</tr>
<tr>
<td>5.12.1 Introduction</td>
<td>1653</td>
</tr>
<tr>
<td>5.12.2 Random Variables</td>
<td>1654</td>
</tr>
<tr>
<td>5.12.3 Sampling Procedures</td>
<td>1655</td>
</tr>
<tr>
<td>5.12.4 The Expected Value Problem</td>
<td>1656</td>
</tr>
<tr>
<td>5.12.5 What types of models can DE handle?</td>
<td>1656</td>
</tr>
<tr>
<td>5.12.6 Reformulation Techniques</td>
<td>1657</td>
</tr>
<tr>
<td>5.12.7 Logfile</td>
<td>1660</td>
</tr>
<tr>
<td>5.12.8 Summary of DE Options</td>
<td>1663</td>
</tr>
<tr>
<td>5.12.9 Detailed Descriptions of DE Options</td>
<td>1664</td>
</tr>
<tr>
<td>5.13 DECIS</td>
<td>1665</td>
</tr>
<tr>
<td>5.13.1 DECIS</td>
<td>1666</td>
</tr>
<tr>
<td>5.13.2 GAMS/DECIS</td>
<td>1670</td>
</tr>
<tr>
<td>5.13.3 Description of GAMS/DECIS Options</td>
<td>1682</td>
</tr>
<tr>
<td>5.13.4 Appendix A - GAMS/DECIS Illustrative Examples</td>
<td>1686</td>
</tr>
<tr>
<td>5.13.5 Appendix B - Error Messages</td>
<td>1690</td>
</tr>
<tr>
<td>5.13.6 DECIS License and Warranty</td>
<td>1692</td>
</tr>
<tr>
<td>5.14 DICOPT</td>
<td>1693</td>
</tr>
<tr>
<td>5.14.1 Introduction</td>
<td>1693</td>
</tr>
<tr>
<td>5.14.2 Requirements</td>
<td>1694</td>
</tr>
<tr>
<td>5.14.3 How to Run a Model with GAMS/DICOPT</td>
<td>1694</td>
</tr>
<tr>
<td>5.14.4 Overview of DICOPT</td>
<td>1694</td>
</tr>
<tr>
<td>5.14.5 The Algorithm</td>
<td>1695</td>
</tr>
<tr>
<td>5.14.6 Modeling</td>
<td>1696</td>
</tr>
<tr>
<td>5.14.7 GAMS Options</td>
<td>1699</td>
</tr>
<tr>
<td>5.14.8 DICOPT Options</td>
<td>1701</td>
</tr>
<tr>
<td>5.14.9 DICOPT Output</td>
<td>1713</td>
</tr>
<tr>
<td>5.14.10 Special Notes</td>
<td>1715</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.23 MILES</td>
<td>2033</td>
</tr>
<tr>
<td>5.23.1 Abstract</td>
<td>2033</td>
</tr>
<tr>
<td>5.23.2 Introduction</td>
<td>2033</td>
</tr>
<tr>
<td>5.23.3 The Newton Algorithm</td>
<td>2034</td>
</tr>
<tr>
<td>5.23.4 Lemke’s Method with Implicit Bounds</td>
<td>2036</td>
</tr>
<tr>
<td>5.23.5 The Options File</td>
<td>2039</td>
</tr>
<tr>
<td>5.23.6 Log File Output</td>
<td>2042</td>
</tr>
<tr>
<td>5.23.7 Status File Output</td>
<td>2043</td>
</tr>
<tr>
<td>5.23.8 Termination Messages</td>
<td>2048</td>
</tr>
<tr>
<td>5.23.9 References</td>
<td>2049</td>
</tr>
<tr>
<td>5.24 MINOS and LINDOGlobal</td>
<td>1962</td>
</tr>
<tr>
<td>5.24.1 Introduction</td>
<td>1962</td>
</tr>
<tr>
<td>5.24.2 Supported nonlinear functions</td>
<td>1963</td>
</tr>
<tr>
<td>5.24.3 Diagnosis of Infeasible or Unbounded Models</td>
<td>1964</td>
</tr>
<tr>
<td>5.24.4 GAMS/LINDO output</td>
<td>1965</td>
</tr>
<tr>
<td>5.24.5 The GAMS/LINDO Options</td>
<td>1966</td>
</tr>
<tr>
<td>5.24.6 Summary of GAMS/Lindo Options</td>
<td>1967</td>
</tr>
<tr>
<td>5.24.7 Detailed Descriptions of GAMS/Lindo Options</td>
<td>1978</td>
</tr>
<tr>
<td>5.24.8 Stochastic Programming (SP) in GAMS/Lindo</td>
<td>2032</td>
</tr>
<tr>
<td>5.25 MILES</td>
<td>2033</td>
</tr>
<tr>
<td>5.25.1 Introduction</td>
<td>2033</td>
</tr>
<tr>
<td>5.25.2 Introduction</td>
<td>2033</td>
</tr>
<tr>
<td>5.25.3 The Newton Algorithm</td>
<td>2034</td>
</tr>
<tr>
<td>5.25.4 Lemke’s Method with Implicit Bounds</td>
<td>2036</td>
</tr>
<tr>
<td>5.25.5 The Options File</td>
<td>2039</td>
</tr>
<tr>
<td>5.25.6 Log File Output</td>
<td>2042</td>
</tr>
<tr>
<td>5.25.7 Status File Output</td>
<td>2043</td>
</tr>
<tr>
<td>5.25.8 Termination Messages</td>
<td>2048</td>
</tr>
<tr>
<td>5.25.9 References</td>
<td>2049</td>
</tr>
<tr>
<td>5.26 MINOS and QUADMINOS</td>
<td>2053</td>
</tr>
<tr>
<td>5.26.1 Introduction</td>
<td>2053</td>
</tr>
<tr>
<td>5.26.2 How to Run a Model with GAMS/MINOS</td>
<td>2054</td>
</tr>
<tr>
<td>5.26.3 Overview of GAMS/MINOS</td>
<td>2054</td>
</tr>
<tr>
<td>5.26.4 Modeling Issues</td>
<td>2058</td>
</tr>
<tr>
<td>5.26.5 GAMS Options</td>
<td>2061</td>
</tr>
<tr>
<td>5.26.6 Summary of MINOS Options</td>
<td>2062</td>
</tr>
<tr>
<td>5.26.7 Special Notes</td>
<td>2064</td>
</tr>
<tr>
<td>5.26.8 The GAMS/MINOS Log File</td>
<td>2065</td>
</tr>
<tr>
<td>5.26.9 Detailed Description of MINOS Options</td>
<td>2070</td>
</tr>
<tr>
<td>5.26.10 Exit Conditions</td>
<td>2084</td>
</tr>
<tr>
<td>5.27 MOSEK</td>
<td>2086</td>
</tr>
<tr>
<td>5.27.1 Introduction</td>
<td>2087</td>
</tr>
<tr>
<td>5.27.2 Solver Options</td>
<td>2089</td>
</tr>
<tr>
<td>5.27.3 The MOSEK Log File</td>
<td>2100</td>
</tr>
<tr>
<td>5.27.4 Semidefinite Programming with GAMS/MOSEK (experimental)</td>
<td>2104</td>
</tr>
<tr>
<td>5.27.5 Detailed Descriptions of MOSEK Options</td>
<td>2109</td>
</tr>
<tr>
<td>5.27.6 Setting up a GAMS/MOSEK-Link license</td>
<td>2138</td>
</tr>
<tr>
<td>5.28 NLPEC</td>
<td>2139</td>
</tr>
<tr>
<td>5.28.1 Introduction</td>
<td>2139</td>
</tr>
<tr>
<td>5.28.2 Usage</td>
<td>2139</td>
</tr>
<tr>
<td>5.28.3 Reformulation</td>
<td>2139</td>
</tr>
<tr>
<td>5.28.4 Options</td>
<td>2147</td>
</tr>
<tr>
<td>5.28.5 Open Architecture</td>
<td>2150</td>
</tr>
<tr>
<td>5.29 Octeract Engine</td>
<td>2151</td>
</tr>
<tr>
<td>5.29.1 Usage</td>
<td>2151</td>
</tr>
<tr>
<td>5.29.2 List of Octeract Options</td>
<td>2152</td>
</tr>
<tr>
<td>5.30 ODHCPLEX</td>
<td>2163</td>
</tr>
<tr>
<td>5.30.1 Introduction</td>
<td>2163</td>
</tr>
<tr>
<td>5.30.2 Specifying Model Structure</td>
<td>2164</td>
</tr>
<tr>
<td>5.30.3 Heuristic Parameters</td>
<td>2165</td>
</tr>
<tr>
<td>5.30.4 Parallel execution using multiple threads</td>
<td>2175</td>
</tr>
</tbody>
</table>
6 Tools Manuals

6.1 Tools Category ......................................................... 2653
  6.1.1 GAMS Integrated Development Environments 2653
  6.1.2 Data Exchange ................................................. 2653
  6.1.3 GDX Service ................................................... 2653
  6.1.4 Data Transformation .......................................... 2654
  6.1.5 Other Tools .................................................... 2654

6.2 List of Tools ....................................................... 2654

6.3 Supported Platforms .............................................. 2656

6.4 ASK ............................................................... 2657
  6.4.1 Usage ........................................................ 2657
  6.4.2 Calling ASK utility from GAMS ............................... 2659
  6.4.3 Radio Button ................................................ 2662
  6.4.4 Combo Box .................................................. 2664
  6.4.5 List and Checklist Box ..................................... 2666
  6.4.6 File Open Dialog Box ...................................... 2667

6.5 CHK4UPD .......................................................... 2668

6.6 CHOLESKY .......................................................... 2669
  6.6.1 Usage ........................................................ 2669
  6.6.2 Example ...................................................... 2669

6.7 CSDP ............................................................... 2670
  6.7.1 Usage ........................................................ 2670

6.8 CSV2GDX ............................................................. 2671
  6.8.1 Usage ........................................................ 2671
  6.8.2 Options ...................................................... 2671
  6.8.3 Advances and limitations .................................. 2674
  6.8.4 Getting Started ............................................. 2675
  6.8.5 Additional Examples for extended Use ...................... 2682

6.9 EIGENVALUE .......................................................... 2686
  6.9.1 Usage ........................................................ 2686
  6.9.2 Example ...................................................... 2686

6.10 EIGENVECTOR ........................................................ 2687
  6.10.1 Usage ........................................................ 2687
  6.10.2 Example ...................................................... 2688

6.11 ENDECRYPT .......................................................... 2689
  6.11.1 Usage ........................................................ 2689

6.12 FINDTHISGAMS ....................................................... 2689
  6.12.1 Introduction ................................................ 2689
  6.12.2 Usage ...................................................... 2690

6.13 GAMS Studio ........................................................ 2691
  6.13.1 Motivation ................................................... 2691
  6.13.2 Central Widgets ............................................. 2691
  6.13.3 Further Studio Widgets .................................... 2718
  6.13.4 MIRO ......................................................... 2737
  6.13.5 NEOS ........................................................ 2737
  6.13.6 GAMS Engine ................................................ 2738
  6.13.7 Dialogs and Actions ....................................... 2739
  6.13.8 Terminal ..................................................... 2749
  6.13.9 Command Line Options ..................................... 2749
  6.13.10 General Shortcuts ........................................ 2750
  6.13.11 Usage Hints ................................................. 2751
  6.13.12 System Requirements ...................................... 2752
  6.13.13 Comparing GAMS Studio and GAMSIDE .................... 2753

6.14 GDX2ACCESS .......................................................... 2755
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.14.1</td>
<td>Overview</td>
<td>2755</td>
</tr>
<tr>
<td>6.14.2</td>
<td>Usage</td>
<td>2756</td>
</tr>
<tr>
<td>6.14.3</td>
<td>Options</td>
<td>2756</td>
</tr>
<tr>
<td>6.14.4</td>
<td>Examples</td>
<td>2758</td>
</tr>
<tr>
<td>6.14.5</td>
<td>References</td>
<td>2763</td>
</tr>
<tr>
<td>6.15</td>
<td>GDX2HAR and HAR2GDX</td>
<td>2764</td>
</tr>
<tr>
<td>6.15.1</td>
<td>Syntax</td>
<td>2764</td>
</tr>
<tr>
<td>6.15.2</td>
<td>GDX2HAR Examples</td>
<td>2766</td>
</tr>
<tr>
<td>6.15.3</td>
<td>HAR2GDX Examples</td>
<td>2771</td>
</tr>
<tr>
<td>6.15.4</td>
<td>Concluding Remarks</td>
<td>2772</td>
</tr>
<tr>
<td>6.16</td>
<td>GDX2SQLITE</td>
<td>2773</td>
</tr>
<tr>
<td>6.16.1</td>
<td>Introduction</td>
<td>2773</td>
</tr>
<tr>
<td>6.16.2</td>
<td>Usage</td>
<td>2773</td>
</tr>
<tr>
<td>6.16.3</td>
<td>How data is stored</td>
<td>2774</td>
</tr>
<tr>
<td>6.16.4</td>
<td>SQLite Browsers and compatible software</td>
<td>2776</td>
</tr>
<tr>
<td>6.17</td>
<td>GDX2VEDA</td>
<td>2778</td>
</tr>
<tr>
<td>6.17.1</td>
<td>Usage</td>
<td>2778</td>
</tr>
<tr>
<td>6.17.2</td>
<td>Examples</td>
<td>2779</td>
</tr>
<tr>
<td>6.17.3</td>
<td>Detailed Help Message</td>
<td>2779</td>
</tr>
<tr>
<td>6.18</td>
<td>GDX2XLS</td>
<td>2781</td>
</tr>
<tr>
<td>6.18.1</td>
<td>Overview</td>
<td>2781</td>
</tr>
<tr>
<td>6.18.2</td>
<td>AutoFilter</td>
<td>2783</td>
</tr>
<tr>
<td>6.18.3</td>
<td>Options</td>
<td>2783</td>
</tr>
<tr>
<td>6.18.4</td>
<td>Examples</td>
<td>2785</td>
</tr>
<tr>
<td>6.19</td>
<td>GDXCOPY</td>
<td>2788</td>
</tr>
<tr>
<td>6.19.1</td>
<td>Usage</td>
<td>2788</td>
</tr>
<tr>
<td>6.19.2</td>
<td>Example</td>
<td>2789</td>
</tr>
<tr>
<td>6.20</td>
<td>GDXDIFF</td>
<td>2790</td>
</tr>
<tr>
<td>6.20.1</td>
<td>Usage</td>
<td>2790</td>
</tr>
<tr>
<td>6.20.2</td>
<td>Options</td>
<td>2790</td>
</tr>
<tr>
<td>6.20.3</td>
<td>Examples</td>
<td>2792</td>
</tr>
<tr>
<td>6.21</td>
<td>GDXDUMP</td>
<td>2793</td>
</tr>
<tr>
<td>6.21.1</td>
<td>Usage</td>
<td>2793</td>
</tr>
<tr>
<td>6.21.2</td>
<td>Options</td>
<td>2793</td>
</tr>
<tr>
<td>6.21.3</td>
<td>Examples</td>
<td>2797</td>
</tr>
<tr>
<td>6.21.4</td>
<td>Adding double Quotes to an user defined Header when writing to CSV</td>
<td>2803</td>
</tr>
<tr>
<td>6.22</td>
<td>GDXMERGE</td>
<td>2803</td>
</tr>
<tr>
<td>6.22.1</td>
<td>Usage</td>
<td>2803</td>
</tr>
<tr>
<td>6.22.2</td>
<td>Options</td>
<td>2804</td>
</tr>
<tr>
<td>6.22.3</td>
<td>Examples</td>
<td>2805</td>
</tr>
<tr>
<td>6.23</td>
<td>GDXMRW</td>
<td>2807</td>
</tr>
<tr>
<td>6.23.1</td>
<td>Introduction</td>
<td>2808</td>
</tr>
<tr>
<td>6.23.2</td>
<td>Data Transfer</td>
<td>2809</td>
</tr>
<tr>
<td>6.23.3</td>
<td>Extended use</td>
<td>2821</td>
</tr>
<tr>
<td>6.23.4</td>
<td>Acknowledgements</td>
<td>2825</td>
</tr>
<tr>
<td>6.23.5</td>
<td>APPENDIX A - Configuring GDXMRW</td>
<td>2825</td>
</tr>
<tr>
<td>6.23.6</td>
<td>APPENDIX B - Utility functions: gdxWhos and gdxInfo</td>
<td>2826</td>
</tr>
<tr>
<td>6.23.7</td>
<td>APPENDIX C - Calling GAMS model from MATLAB</td>
<td>2827</td>
</tr>
<tr>
<td>6.24</td>
<td>GDXRANK</td>
<td>2832</td>
</tr>
<tr>
<td>6.24.1</td>
<td>Usage</td>
<td>2832</td>
</tr>
<tr>
<td>6.24.2</td>
<td>General Comments</td>
<td>2833</td>
</tr>
<tr>
<td>6.24.3</td>
<td>GDXRANK Example</td>
<td>2833</td>
</tr>
<tr>
<td>6.24.4</td>
<td>RANK Example</td>
<td>2834</td>
</tr>
<tr>
<td>6.25</td>
<td>GDXRENAME</td>
<td>2841</td>
</tr>
<tr>
<td>6.25.1</td>
<td>Usage</td>
<td>2841</td>
</tr>
<tr>
<td>6.25.2</td>
<td>Options</td>
<td>2842</td>
</tr>
<tr>
<td>6.25.3</td>
<td>Example</td>
<td>2842</td>
</tr>
<tr>
<td>6.26</td>
<td>GDXRRW</td>
<td>2843</td>
</tr>
</tbody>
</table>
## Application Programming Interfaces

### 7 Object-oriented APIs

- 7.1 Reference Manuals
- 7.2 Tutorials
- 7.3 Examples
- 7.4 Release Notes
- 7.5 Supported Platforms

### 7.6 Expert-Level APIs

- 7.6.1 Supported Platforms
- 7.6.2 GAMS Transfer Matlab

### 7.7 C++ API

- 7.7.1 GAMS Transfer R

### 7.8 Tutorial

- 7.8.1 Getting Started
- 7.8.2 Important Classes of the API
- 7.8.3 How to use API

### 7.9 Tutorial

- 7.9.1 Getting Started
- 7.9.2 Important Classes of the API
- 7.9.3 How to use API
- 7.9.4 How to use the pre configured example projects

### 7.10 Tutorial

- 7.10.1 Getting started
- 7.10.2 Important Classes of the API
- 7.10.3 How to use API

### 7.11 Tutorial

- 7.11.1 Getting started
- 7.11.2 Important Classes of the API
- 7.11.3 How to use the API

### 7.12 Tutorial

- 7.12.1 Getting started
- 7.12.2 Important Classes of the API
- 7.12.3 How to use the API

### 7.13 Release Notes

- 7.13.1 39.1.0
- 7.13.2 35.1.0
- 7.13.3 32.1.0
- 7.13.4 31.1.0
- 7.13.5 30.1.0
- 7.13.6 29.1.0
- 7.13.7 28.1.0
- 7.13.8 27.1.0
- 7.13.9 26.1.0
- 7.13.10 25.1.0
- 7.13.11 24.1.0
- 7.13.12 23.1.0
- 7.13.13 22.1.0
- 7.13.14 21.1.0
- 7.13.15 20.1.0
- 7.13.16 19.1.0
- 7.13.17 18.1.0
- 7.13.18 17.1.0
- 7.13.19 16.1.0
- 7.13.20 15.1.0
- 7.13.21 14.1.0
- 7.13.22 13.1.0
- 7.13.23 12.1.0
- 7.13.24 11.1.0
- 7.13.25 10.1.0
- 7.13.26 9.1.0
- 7.13.27 8.1.0
- 7.13.28 7.1.0
- 7.13.29 6.1.0
- 7.13.30 5.1.0
- 7.13.31 4.1.0
- 7.13.32 3.1.0
- 7.13.33 2.1.0
- 7.13.34 1.1.0
- 7.13.35 0.1.0
- 7.13.36 -1.0

### 7.14 Example

- 7.14.1 Usage
- 7.14.2 Example

### 7.15 Example

- 7.15.1 Usage
- 7.15.2 Example

### 7.16 Example

- 7.16.1 Usage
- 7.16.2 Example

### 7.17 Example

- 7.17.1 Usage
- 7.17.2 Example

### 7.18 Example

- 7.18.1 Usage
- 7.18.2 Example

### 7.19 Example

- 7.19.1 Usage
- 7.19.2 Example

### 7.20 Example

- 7.20.1 Usage
- 7.20.2 Example

### 7.21 Example

- 7.21.1 Usage
- 7.21.2 Example

### 7.22 Example

- 7.22.1 Usage
- 7.22.2 Example

### 7.23 Example

- 7.23.1 Usage
- 7.23.2 Example

### 7.24 Example

- 7.24.1 Usage
- 7.24.2 Example

### 7.25 Example

- 7.25.1 Usage
- 7.25.2 Example

### 7.26 Example

- 7.26.1 Usage
- 7.26.2 Example

### 7.27 Example

- 7.27.1 Usage
- 7.27.2 Example

### 7.28 Example

- 7.28.1 Usage
- 7.28.2 Example

### 7.29 Example

- 7.29.1 Usage
- 7.29.2 Example

### 7.30 Example

- 7.30.1 Usage
- 7.30.2 Example

### 7.31 Example

- 7.31.1 Usage
- 7.31.2 Example

### 7.32 Example

- 7.32.1 Usage
- 7.32.2 Example

### 7.33 Example

- 7.33.1 Usage
- 7.33.2 Example

### 7.34 Example

- 7.34.1 Usage
- 7.34.2 Example

### 7.35 Example

- 7.35.1 Usage
- 7.35.2 Example

### 7.36 Example

- 7.36.1 Usage
- 7.36.2 Example

### 7.37 Example

- 7.37.1 Usage
- 7.37.2 Example

### 7.38 Example

- 7.38.1 Usage
- 7.38.2 Example

### 7.39 Example

- 7.39.1 Usage
- 7.39.2 Example

### 7.40 Example

- 7.40.1 Usage
- 7.40.2 Example

### 7.41 Example

- 7.41.1 Usage
- 7.41.2 Example

### 7.42 Example

- 7.42.1 Usage
- 7.42.2 Example

### 7.43 Example

- 7.43.1 Usage
- 7.43.2 Example

### 7.44 Example

- 7.44.1 Usage
- 7.44.2 Example
# Contents

7.14 Release Notes .......................... 3248
  7.14.1 39.1.0 (April 2022) .................. 3248
  7.14.2 38.1.0 (April 2021) ................. 3248
  7.14.3 29.1.0 (July 2020) .................. 3248
  7.14.4 28.1.0 (November 2019) ............. 3248
  7.14.5 28.0.1 (August 2019) ............... 3248
  7.14.6 27.1.1 (May 2018) .................. 3249
  7.14.7 25.0.1 (January 2018) .............. 3249
  7.14.8 24.8.1 (December 2016) ............ 3249
  7.14.9 24.7.4 (September 2016) .......... 3249
  7.14.10 24.7.1 (March 2016) .............. 3250
  7.14.11 24.5.1 (August 2015) ............. 3250
  7.14.12 24.4.2 (March 2015) .............. 3250
  7.14.13 24.4.1 (December 2014) .......... 3250
  7.14.14 24.3.3 (September 2014) ......... 3250
  7.14.15 24.3.2 (August 2014) ............ 3251
  7.14.16 24.3.1 (July 2014) ............... 3251
  7.14.17 24.2.3 (May 2014) ............... 3252
  7.14.18 24.2.2 (March 2014) ............. 3253
  7.14.19 24.1.3 (July 2013) ............... 3254
  7.14.20 24.1.2 (December 2013) ......... 3254
  7.14.21 24.1.1 (May 2013) ................ 3256
  7.14.22 24.1.0 (February 2013) .......... 3257
  7.14.23 24.0.1 (December 2012) ......... 3257

7.15 Release Notes .......................... 3257
  7.15.1 40.1.0 ............................ 3257
  7.15.2 39.1.0 ............................ 3257
  7.15.3 38.1.0 ............................ 3257
  7.15.4 37.1.0 ............................ 3257
  7.15.5 34.1.0 ............................ 3257
  7.15.6 32.1.0 ............................ 3258
  7.15.7 31.1.0 ............................ 3258
  7.15.8 30.1.0 ............................ 3258
  7.15.9 29.1.0 ............................ 3258
  7.15.10 28.1.0 ............................ 3258
  7.15.11 25.0.1 ............................ 3259
  7.15.12 24.9.2 ............................ 3259
  7.15.13 24.9.1 ............................ 3259
  7.15.14 24.8.4 ............................ 3259
  7.15.15 24.8.1 ............................ 3259
  7.15.16 24.7.4 ............................ 3259
  7.15.17 24.7.2 ............................ 3259
  7.15.18 24.7.1 ............................ 3260
Index

Bibliography
Chapter 1

GAMS Documentation Center

The GAMS Documentation Center provides you with the technical information on getting started, using and maintaining our GAMS (General Algebraic Modeling System) products.

- Preface
- Release Notes - 40.1.0 Major release (August 01, 2022)
- User’s Guide
  - Installation and Licensing - Guides on installing GAMS for various platforms and using a GAMS license
  - Tutorials and Examples - Step-by-step guides including a number of examples
  - GAMS Language and Environment - Guide through components of GAMS Language and the environment for executing a GAMS model
- Solver Manuals - Manuals of solvers available in the distribution
- Tools Manuals - Manuals of tools available in the distribution
- Application Programming Interfaces - Manuals of Application Programming Interfaces
- Glossary - An alphabetical list of GAMS terms.
- Bibliography

1.1 Model Libraries

From the early stages of the development of GAMS we have collected models to be used in libraries of examples. Many of these are standard textbook examples and can be used in classes on problem formulation or to illustrate points about GAMS. Others are models that have been used in policy or sector analysis and are interesting for both the methods and the data they use. These model libraries are included with all GAMS systems and are also available online.

The following model libraries are available:

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.
• **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.

• **GAMS Data Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.

• **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.

• **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.

• **FIN Library** - includes GAMS practical financial optimization models described in the book *Practical Financial Optimization: Decision Making for Financial Engineers* by Consiglio, Nielsen and Zenios,

• **NOA Library** - includes GAMS nonlinear optimization applications models based on the book *Nonlinear Optimization Applications Using the GAMS Technology* by Neculai Andrei.

• **PSOPT Library** - includes GAMS optimization models based on the book *Power System Optimization Modelling in GAMS* by by Alireza Soroudi.

See [Accessing Model Libraries](#) on how to access a GAMS model from the model libraries.

For large parts of the documentation, references to models from the model libraries are enclosed in square parenthesis (for example, *[TRNSPORT]*).

### 1.2 Further Help

If you have a further question which is not answered by the documentation above, you can get further help from our [GAMS-FAQ](#) which contains materials collected from various support activities or post your question to [GAMS World Forum](#). You can also sign up to our [Newsletters](#) to get the latest information from GAMS. There is also [GAMS Lessons](#), a YouTube Video Channel providing you with some tutorials on how to use our system.

There are a number of contributed documentations that have been contributed by GAMS users as well as presentations, books, posters, and advertisements contributed by people working with GAMS. The McCarl GAMS User Guide are available in various formats (PDF and HTML) and can also be found in the GAMS distribution under mccarl/ subdirectory with an incremental installer (ZIP) available for download.

You can also visit our upcoming courses and workshops or contact one of solution specialists who have wide experience in GAMS modeling.

If you experience a problem using GAMS please contact our [Technical Support](#). If you have a confidential model that you cannot share as is, please read the information on conversion of models before submitting your models to our technical support.
Chapter 2

Preface

The GAMS documentation has changed considerably since it was first released in 1988: in format, in scope and content, and in how it is distributed and accessed. At the same time, it maintains many elements of the original style: part tutorial, part user’s guide, and part reference manual. We describe here the evolution of the GAMS documentation and related topics.

The first published GAMS documentation is the book *GAMS: A User’s Guide* (aka the “red book”) by Brooke, Kendrick, and Meeraus, published in 1988 by The Scientific Press. Copies were sold in bookstores and included a chapter on the model library (100 models at that time), a much-appreciated and well-used index, appendices describing the solvers available (MINOS and ZOOM), and a 5.25 inch floppy disk containing a student version of "PC-GAMS version 2.05". This was followed up in 1992 with *GAMS: A User’s Guide, Release 2.25* (aka the "blue book") by a different publisher (Boyd & Fraser), a soft cover, and no floppy disk: by this time the software was available directly from GAMS. The final section of installation notes in the red book was replaced in the blue book by notes on the new features in GAMS 2.25, but the content was otherwise unchanged.

As the pace of development increased, it became clear that we needed a new documentation scheme, one that would allow us to update the document continuously as language features, models and model libraries, solvers, and tools/utilities were added to or updated in the system. Consequently, we converted the existing documentation to MS Word, expanded, updated, and edited it as appropriate, and sent it to the copy shop to be printed. This was a group effort spearheaded by Ramesh Raman, whose name was included as an author in the User’s Guide first included (along with a CD) in shipments of our 18.0 release in Feb 1999. The solver manuals in this release were collected from various sources (with varying styles, content, and organization) and put into a separate binder. A further push was required to unify this solver documentation (using LaTeX) to use a common style and organization: this unified Solver Manual was included with the 21.0 release of May 2003. In addition to hard-copy, all of these documents were included as PDF in the GAMS system and freely available via the Web. Thus, the shift away from published documentation to manuals produced in-house went hand-in-hand with a shift from printed documents to viewing and searching PDF documents.
During this time Bruce McCarl had been regularly teaching GAMS courses: basic and advanced, targeting a general audience but with a distinct flavor of agricultural economics. As part of this he developed course material to use in his classes, and he was in a good position to notice that the GAMS User Guide was not keeping up with all the updates to the software. With some encouragement from GAMS, Bruce started the ambitious project of creating a new GAMS User’s Guide, one that would be more complete, more to his taste with an increase in tutorial flavor, and designed to take full advantage of being a Web document: linked, cross-referenced, searchable, and indexed. The end result was the McCarl Guide that emerged as a PDF based on a Word document in 2002. This was redone in the more convenient CHM format with the 22.1 release of Mar 2006 and renamed as the Expanded GAMS Guide (McCarl) in release 22.3 of Nov 2006. We also kept the original GAMS User’s Guide, by this time as PDF generated from LaTeX source. With its improved content, organization, and navigability, Bruce's new guide was a boon to all GAMS users, not just those taking his courses. We owe a debt of gratitude to Bruce for this and many other contributions.

The Expanded GAMS Guide (McCarl) was never intended to be printed and was not shipped as hard-copy with GAMS. In 2013, GAMS stopped making physical shipments altogether and moved to an online order-fulfillment system. For those who still wanted printed documentation, the documents were also made available via Amazon’s CreateSpace, but most users browsed the documents online or as part of an installed GAMS system, often via the GAMS IDE, so the need for printable documents was small and growing smaller.

Having two user's guides was counter-intuitive or confusing for some users, especially when the content for some topics was different, missing, or even contradictory. The additional effort required to maintain two documents became more noticeable, especially since the Expanded User Guide (McCarl) was implemented using a Windows-only product that did not lend itself to version control via SVN or git. When we documented our object-oriented APIs in Feb 2013, we used the opportunity to experiment with a doxygen-based authoring process that uses text files as source, is platform-independent, and produces output in multiple formats. The results were good, and in 2015 we hired a consultant, Martha Loewe, to research what it would take to convert all of the GAMS documentation to use the doxygen-based process and to set up a framework for this. We eventually committed fully to this process and produced a unified set of documentation as of release 24.9 in Aug 2017. This involved reorganizing things somewhat to better fit a linked document, merging the two user guides into one (while trying to maintain the best qualities of each), adding links throughout, and submitting everything to review and editing by internal subject matter experts so the content is accurate, updated, and complete. The end result is something we believe is an improvement over its predecessors, but it is also something we are committed to maintaining and improving. The current system allows and encourages immediate updates by the whole GAMS team and has helped foster a culture and attitude where this takes place.

### Table 2.1: Timeline for documentation released with GAMS software

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Month/Year</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>Feb-99</td>
<td>IDE released plus IDE help (hlp), no other documentation</td>
</tr>
<tr>
<td>19.0</td>
<td>Jan-00</td>
<td>GAMS Users Guide (pdf from Word), installation notes, plus some solver chapters</td>
</tr>
<tr>
<td>19.1</td>
<td>Mar-00</td>
<td>Integration of documentation in the IDE</td>
</tr>
<tr>
<td>21.0</td>
<td>May-03</td>
<td>McCarl Guide (collection of linked pdf documents) on installation CD and web plus instructions how to integrate into GAMS system, IDE help format change hlp-&gt;chm, other tools document gdxutils.pdf, combined allsolvers.pdf plus individual solver chapters (with TOC document)</td>
</tr>
<tr>
<td>21.3</td>
<td>Jan-04</td>
<td>Added gdxutils.chm</td>
</tr>
<tr>
<td>21.4</td>
<td>Feb-04</td>
<td>Integrate the McCarl Guide</td>
</tr>
<tr>
<td>22.0</td>
<td>Aug-05</td>
<td>Added releasenotes.htm</td>
</tr>
<tr>
<td>22.1</td>
<td>Mar-06</td>
<td>McCarl Guide as chm, GAMS Users Guide (pdf from LaTeX)</td>
</tr>
</tbody>
</table>
### Timeline for documentation released with GAMS software

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Month/Year</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.3</td>
<td>Nov-06</td>
<td>McCarl Guide renamed to Expanded GAMS Guide (McCarl)</td>
</tr>
<tr>
<td>22.5</td>
<td>Jun-07</td>
<td>Tools documentation as chm and pdf</td>
</tr>
<tr>
<td>24.0</td>
<td>Feb-13</td>
<td>API documentation (.NET as chm, others as pdf)</td>
</tr>
<tr>
<td>24.1</td>
<td>Jul-13</td>
<td>.NET API tutorial as chm</td>
</tr>
<tr>
<td>24.2</td>
<td>Dec-13</td>
<td>Other API tutorials as pdf</td>
</tr>
<tr>
<td>24.3</td>
<td>Jul-14</td>
<td>API documentation as html</td>
</tr>
<tr>
<td>24.4</td>
<td>Dec-14</td>
<td>Solvers and many tools as html</td>
</tr>
<tr>
<td>24.5</td>
<td>Sep-15</td>
<td>GAMS Users Guide as chm, pdf, and html</td>
</tr>
<tr>
<td>24.8</td>
<td>Dec-16</td>
<td>Drop chm of GAMS Users Guide, html based unified documentation</td>
</tr>
<tr>
<td>25.1</td>
<td>May-18</td>
<td>Documentation integration into studio</td>
</tr>
</tbody>
</table>
Chapter 3

Release Notes

3.1 Release Types

**Major releases** contain substantial changes to the GAMS system. The License Check date is set to the release date of the major release.

**Minor releases** are mainly issued to provide bug fixes, performance improvements, and maintenance releases of solver libraries. Additionally, they can provide a few new features that do not change existing behavior. The License Check Date remains the same as for the prior major release. This means that any license file that worked with the prior major release will also work with this minor release.

**Maintenance releases** do not provide any new features. They are issued to provide bug fixes, performance improvements, and maintenance releases of solver libraries. The License Check Date remains the same as for the prior major release. This means that any license file that worked with the prior major release will also work with this maintenance release.

3.2 Release History

- 40 Distribution  (distribution start page)
  - 40.1.0 Major release (August 01, 2022)

- 39 Distribution  (distribution start page)
  - 39.1.0 Major release (May 03, 2022)
  - 39.1.1 Maintenance release (May 18, 2022)
  - 39.2.0 Minor release (June 02, 2022)
  - 39.2.1 Maintenance release (June 21, 2022)
  - 39.3.0 Minor release (July 07, 2022)

- 38 Distribution  (distribution start page)
  - 38.1.0 Major release (January 31, 2022)
  - 38.2.0 Minor release (February 17, 2022)
  - 38.2.1 Maintenance release (February 19, 2022)
  - 38.3.0 Minor release (April 05, 2022)

- 37 Distribution  (distribution start page)
- 37.1.0 Major release (November 11, 2021)

- **36 Distribution** (distribution start page)
  - 36.1.0 Major release (August 02, 2021)
  - 36.2.0 Minor release (September 03, 2021)

- **35 Distribution** (distribution start page)
  - 35.1.0 Major release (April 29, 2021)
  - 35.2.0 Minor release (June 02, 2021)

- **34 Distribution** (distribution start page)
  - 34.1.0 Major release (January 29, 2021)
  - 34.2.0 Minor release (February 05, 2021)
  - 34.3.0 Minor release (February 25, 2021)

- **33 Distribution** (distribution start page)
  - 33.1.0 Major release (November 01, 2020)
  - 33.2.0 Minor release (December 01, 2020)

- **32 Distribution** (distribution start page)
  - 32.1.0 Major release (July 31, 2020)
  - 32.2.0 Minor release (August 26, 2020)

- **31 Distribution** (distribution start page)
  - 31.1.0 Major release (May 01, 2020)
  - 31.1.1 Maintenance release (May 16, 2020)
  - 31.2.0 Minor release (June 19, 2020)

- **30 Distribution** (distribution start page)
  - 30.1.0 Major release (January 10, 2020)
  - 30.2.0 Minor release (February 07, 2020)
  - 30.3.0 Minor release (March 06, 2020)

- **29 Distribution** (distribution start page)
  - 29.1.0 Major release (November 15, 2019)

- **28 Distribution** (distribution start page)
  - 28.1.0 Major release (August 02, 2019)
  - 28.2.0 Minor release (August 19, 2019)

- **27 Distribution** (distribution start page)
  - 27.1.0 Major release (April 24, 2019)
  - 27.2.0 Minor release (May 23, 2019)
  - 27.3.0 Minor release (July 04, 2019)

- **26 Distribution** (distribution start page)
  - 26.1.0 Major release (February 02, 2019)

- **25.1 Distribution** (distribution start page)
  - 25.1.1 Major release (May 19, 2018)
  - 25.1.2 Minor release (August 01, 2018)
3.2 Release History

- 25.1.3 Minor release (October 30, 2018)

**25.0 Distribution**
- 25.0.1 Major release (January 17, 2018)
- 25.0.2 Maintenance release (January 31, 2018)
- 25.0.3 Minor release (March 21, 2018)

**24.9 Distribution**
- 24.9.1 Major release (August 30, 2017)
- 24.9.2 Minor release (November 14, 2017)

**24.8 Distribution**
- 24.8.1 Major release (December 21, 2016)
- 24.8.2 Maintenance release (January 03, 2017)
- 24.8.3 Minor release (January 28, 2017)
- 24.8.4 Minor release (April 10, 2017)
- 24.8.5 Maintenance release (May 10, 2017)

**24.7 Distribution**
- 24.7.1 Major release (March 14, 2016)
- 24.7.2 Minor release (July 07, 2016)
- 24.7.3 Maintenance release (July 11, 2016)
- 24.7.4 Minor release (September 19, 2016)

**24.6 Distribution**
- 24.6.1 Major release (January 18, 2016)

**24.5 Distribution**
- 24.5.1 Major release (September 23, 2015)
- 24.5.2 Maintenance release (September 29, 2015)
- 24.5.3 Maintenance release (October 01, 2015)
- 24.5.4 Maintenance release (October 15, 2015)
- 24.5.5 Maintenance release (November 25, 2015)
- 24.5.6 Maintenance release (November 27, 2015)

**24.4 Distribution**
- 24.4.1 Major release (December 20, 2014)
- 24.4.2 Minor release (March 15, 2015)
- 24.4.3 Maintenance release (April 02, 2015)
- 24.4.4 Maintenance release (May 12, 2015)
- 24.4.5 Maintenance release (May 26, 2015)
- 24.4.6 Minor release (June 26, 2015)

**24.3 Distribution**
- 24.3.1 Major release (July 31, 2014)
- 24.3.2 Minor release (August 29, 2014)
- 24.3.3 Minor release (September 19, 2014)

**24.2 Distribution**
– 24.2.1 Major release (December 09, 2013)
– 24.2.2 Maintenance release (March 04, 2014)
– 24.2.3 Maintenance release (May 22, 2014)

• 24.1 Distribution
  – 24.1.1 Major release (May 30, 2013)
  – 24.1.2 Maintenance release (June 16, 2013)
  – 24.1.3 Maintenance release (July 26, 2013)

• 24.0 Distribution
  – 24.0.1 Major release (December 24, 2012)
  – 24.0.2 Maintenance release (February 14, 2013)

• 23.9 Distribution
  – 23.9.1 Major release (July 04, 2012)
  – 23.9.2 Maintenance release (August 29, 2012)
  – 23.9.3 Maintenance release (September 26, 2012)
  – 23.9.4 Maintenance release (October 20, 2012)
  – 23.9.5 Maintenance release (November 09, 2012)

• 23.8 Distribution
  – 23.8.1 Major release (March 17, 2012)
  – 23.8.2 Maintenance release (April 05, 2012)

• 23.7 Distribution
  – 23.7.1 Major release (July 14, 2011)
  – 23.7.2 Maintenance release (July 22, 2011)
  – 23.7.3 Maintenance release (August 23, 2011)

• 23.6 Distribution
  – 23.6.2 Major release (December 13, 2010)
  – 23.6.3 Maintenance release (February 15, 2011)
  – 23.6.4 Maintenance release (April 01, 2011)
  – 23.6.5 Maintenance release (April 08, 2011)

• 23.5 Distribution
  – 23.5.1 Major release (July 05, 2010)
  – 23.5.2 Maintenance release (August 18, 2010)

• 23.4 Distribution
  – 23.4.1 Major release (May 21, 2010)
  – 23.4.3 Maintenance release (May 24, 2010)

• 23.3 Distribution
  – 23.3.1 Major release (November 01, 2009)
  – 23.3.2 Maintenance release (November 18, 2009)
  – 23.3.3 Maintenance release (December 17, 2009)

• 23.2 Distribution
  – 23.2.1 Minor release (August 14, 2009)
• 23.1 Distribution
  – 23.1.1 Major release (July 13, 2009)
  – 23.1.2 Maintenance release (July 23, 2009)
• 23.0 Distribution
  – 23.0.2 Major release (February 14, 2009)
• 22.9 Distribution
  – 22.9.2 Major release (December 01, 2008)
• 22.8 Distribution
  – 22.8.1 Major release (August 01, 2008)
• 22.7 Distribution
  – 22.7.1 Major release (May 01, 2008)
  – 22.7.2 Maintenance release (May 13, 2008)
• 22.6 Major release (December 24, 2007)
• 22.5 Major release (June 01, 2007)
• 22.4 Major release (February 12, 2007)
• 22.3 Major release (November 27, 2006)
• 22.2 Minor release (April 21, 2006)
• 22.1 Major release (March 15, 2006)
• 22.0 Major release (August 01, 2005)
• 21.7 Major release (April 01, 2005)
• 21.6 Minor release (January 26, 2005)
• 21.5 Minor release (November 11, 2004)
• 21.4 Major release (September 06, 2004)
• 21.3 Major release (January 19, 2004)
• 21.2 Maintenance release (September 03, 2003)
• 21.1 Maintenance release (June 02, 2003)
• 21.0 Major release (May 15, 2003)
• 20.7 Maintenance release (June 14, 2002)
• 20.6 Major release (May 25, 2002)
• 20.5 Maintenance release (January 28, 2002)
• 20.4 Maintenance release (January 21, 2002)
• 20.3 Major release (December 24, 2001)
• 20.2 Maintenance release (November 22, 2001)
• 20.1 Major release (October 31, 2001)
3.3 40 Distribution

3.3.1 40.1.0 Major release (August 01, 2022)

3.3.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Daniel Dias, and Scott McDonald.

3.3.1.2 GAMS System

3.3.1.2.1 GAMS

- Added two new indexed operators sAnd and sOr.
- The file containing the collection of legacy default values has been renamed from gams31config.yaml to gamsLegacyConfig.yaml, as legacy behavior is not entirely identical to GAMS 31.
- Changed default of command line parameter digit to off. As a result, extremely large or small constants in the GAMS source will not result in compilation errors but rather in special values (e.g. 1e300 becomes UNDF, 1e-300 becomes EPS). The old default digit=on has been added to gamsLegacyConfig.yaml.
- Changed default of model attribute tolProj to 0. This means that GAMS no longer by default modifies solution values returned by the solver that are close to variable bounds or equation sides.
- Throw a compilation error consistently, when one tries to load data into an Alias. In the past, this was just ignored in some cases. It appears that when writing to an alias in embedded code an exception is made and GAMS implicitly imports into a set, not an alias. Details about this subtlety can be found in the embedded code chapter.
- Made command line parameter IDCProtect available as runtime option as well (the runtime option gets initialized by the command line parameter).
- Do not allow to load a symbol implicitly from $offEmbeddedCode under $onExternalInput, when it was not declared as external input symbol (the same check was so far done for explicit loading only).
- Changed the interaction of external input and $gdxIn to allow the continued use of a $gdxIn file over multiple external input sections. So, the following code works now, but threw an error in the past:

```plaintext
$onExternalInput
$gdxIn transport.gdx
Set i(*) canning plants ;
$load i
$offExternalInput

* Do something but not opening another GDX file

$onExternalInput
Set j(*) markets ;
$load j
$offExternalInput
```
• Changed the behavior when encountering a symbol with unknown dimension with an execution time load or embedded code statement. In the past, this resulted in an error, now the dimension of that symbol is set to 0 (or 1, if it is a set) implicitly. As a result, the following code runs without error now:

```bash
$call gams transport gdx=default
Variable z;
execute_load 'transport.gdx', z;
display z.l;
```

• Also changed the behavior when encountering a symbol with unknown dimension with a compile time embedded code statement. In the past, this resulted in an error, now the dimension of that symbol is set to the dimension defined in the embedded code block.

• Make sure that we do not change the loop control index by implicit loading from endEmbeddedCode (the same check was so far done for explicit loading only).

• Added an explicit error check to ensure that the command line parameter restartNamed is used together with the command line parameter restart only.

• Changed the put utility facility to ignore the extra double quotes added when the print control is set to 4, 5, or 6. In the past that lead to an error since the commands were not recognized when they are quoted.

• We plan to remove language elements to access some commercial and licensing information with the next major release:
  – System suffixes: components, clipCodes, clipComponentMap, componentSolverMap, gamsLicenses, gamsLicenseTypes, lice1, lice2, licenseEvalN, licenseDateEvalS, licenseDateMaintN, licenseDateMaintS, licenseDateS, licenseDateSysN, licenseDateSysS, licenseDaysEval, licenseDaysMaint, licenseDC, licenseEval, licenseID, licenseInstitution, licenseLevel, licenseLevelText, licenseLicensee, licenseMudText, licensePlatform, licensePlatformText, licenseStatus, licenseStatusText, licenseType, licenseVendor, and vendors.
  – Predefined sets componentEDate, componentMDate, and componentLicenseOptions.
  – Many of these are used in the GAMS Model Library model licememo, which we also plan to remove.

• Fixed that exceeding the internal limit of 2.1 billion instructions did not raise an error.

• Fixed a rare problem which stopped GAMS from terminating if a solver invoked in an asynchronous thread did not return.

• Fixed an error causing missing domain information, when implicit sets are used while creating a dump file.

### 3.3.1.2.2 GMSPython

• Added the packages sqlalchemy and greenlet to GMSPython. On Windows, also added the packages sqlalchemy-access, pyodbc, and pywintypes.

### 3.3.1.2.3 Connect

• Domain information is copied into the Connect database when using agents GAMSReader and GDXReader. Agent GDXWriter exports the domain information to the GDX file.
• Added attribute `duplicateRecords` to `GAMSWriter` and `GDXWriter` to control the way to deal with duplicate records in a Connect container symbol when writing to GAMS or GDX. Note that both agents currently deal with duplicate records in a case sensitive way.

• Added new `DomainWriter` agent to modify domain information of Connect symbols.

• Added new `PandasSQLReader` and `PandasSQLWriter` agents to read from and write to SQL databases.

• Updated `Projection` agent:
  – Changed boolean attribute `dropText` to a string attribute `text`.
  – Added boolean attribute `asParameter` to turn variables into parameters with an extra index to store the suffixes.
  – Allow a list of symbol names for attribute `name` to combine multiple scalar symbols into a single one-dimensional symbol.

• Symbol names in Connect are now case sensitive by default. So casing in the YAML file is important and needs to match e.g. when reading from GDX or GAMS (which use case in-sensitive symbol name lookup) or across agents. There is now a global attribute `caseSensitiveLookup` in agent `Options` to control the case-(in)sensitive symbol lookup.

• Added new attribute `debug` in agent `Options` to provide traceback information in case of an exception.

• Dropped support for writing variables and equations with `CSVWriter` and `PandasExcelWriter`. Variables and equations can be turned into parameters using the `Projection` agent.

• Added attribute `trace` to `GAMSReader`, `GAMSWriter`, `GDXReader`, and `GDXWriter`.

• Fixed bug where `PandasExcelWriter` failed when writing no data.

### 3.3.1.2.4 Embedded Python Code Facility

• Allow to specify source code encoding by providing a comment of format `# coding=<encoding name>` as first line in an embedded Python code section.

• If `gams_.debug` is set to an integer greater than 0 a traceback is printed to `stdout` in case of an exception.

### 3.3.1.2.5 Installer

• For Windows, updated the text in the file association dialog to make clear that the GAMS IDE is deprecated.

### 3.3.1.3 Solvers

#### 3.3.1.3.1 BARON

• New libraries 22.7.23.
  – Improved presolve.

#### 3.3.1.3.2 Bonmin and BonminH

• As announced, the solvers Bonmin and BonminH have been removed. For the time being, libraries of GAMS/Bonmin are available at the [COIN-OR GAMSlinks project](https://www.coin-or.org/GAMSlinks). Note, that GAMS does not test or offer support for these libraries.
3.3.1.3.3 CBC, SCIP, SHOT

- New libraries Cbc 2.10.8, Cgl 0.60.0, Clp 1.17.7, Osi 0.108.7, CoinUtils 2.11.6, nauty 2.7r3.

3.3.1.3.4 CONVERT

- Export prettier formulas (x for \(x^{*1}\), power(x,1), vcpower(x,1) or rpower(x,1)).

3.3.1.3.5 COPT

- New libraries 5.0.2.
  - Added possibility to compute feasible relaxation for infeasible problem. New options FeasRelax and FeasRelaxMode to enable this feature and set relaxation mode, respectively. New dot option .feaspref to specify preferences on relaxation of variable bounds and linear constraint sides.
  - Added options MipStartMode and MipStartNodeLimit to adjust handling of complete and partial MIP starts.
  - Added option ReqFarkasRay.
  - Option crossover has been made available.
- Changed handling of MIP starting point:
  - GAMS/COPT will pass on a partial instead of a full MIP start also if MipStartMode = 2 is selected.
  - When setting the value for a discrete variable in a partial MIP start (tryint > 0), it will be rounded to an integer value.
- For an unbounded LP, a primal ray is now returned in the variables level values and UNBND markers are set in the solution listing.
- For an infeasible LP, a Farkas proof is now returned in the equations marginal values and INFES markers are set in the solution listing.

3.3.1.3.6 GLOMIQO

- As announced, dropped GLOMIQO. The use of ANTIGONE is equivalent.

3.3.1.3.7 GUROBI

- New libraries 9.5.2.
  - Changed aggregate option type from bool to int. Allowed values are 0 (off), 1 (moderate) and 2 (aggressive).
- Added support of certain nonlinear constraints: max, min, smax, smin, and, or, sand, sor, abs, exp, **, log, log2, log10, sin, cos, tan, edist, poly. To make these accessible, Gurobi can now be used for model types (D)NLP and (R)MINLP as well, but equations need to follow a special form.
- Added option multimipstart: Use multiple (partial) mipstarts provided via GDX files.
3.3.1.3.8 HiGHS

- New LP/MIP solver HiGHS 1.2.2 developed by the Edinburgh Research Group in Optimization. HiGHS is included in the GAMS base system for users with an academic GAMS license. Users with a commercial GAMS license need to contact sales@gams.com to enable the use of HiGHS.

3.3.1.3.9 Ipopt

  - Added options ma27_print_level, ma57_print_level, and mumps_print_level.
  - Fixed that variable names for variable bounds in detailed log output were wrong for problems with fixed variables.
- New libraries Mumps 5.5.0.

3.3.1.3.10 KNITRO

- New libraries 13.1.0.
  - Significant performance improvements when using the BFGS/LBFGS Hessian approximation.
  - New options:
    * findiff_estnoise: Estimate of the noise in the model when using finite-difference.
    * bar_mpec_heuristic: Enable heuristic approach when solving MPEC models with the barrier algorithm.
    * mip_heuristic_misqp: Enable the MIP MISQP heuristic.
    * mip_restart: Enable the MIP restart procedure.
    * conic_numthreads: Number of threads to use for operations in the conic algorithm.

3.3.1.3.11 LGO

- As announced, dropped LGO.

3.3.1.3.12 MPSGE

- Increased the internal MAXFUN limit (from 10000 to 50000) on the number of components (inputs, outputs, taxes, etc.) in any MPSGE row.
- Print info about the size of the largest MPSGE function (i.e. the one with the most input/output/tax/etc coefficients) to the sysout file.

3.3.1.3.13 MSNLP

- As announced, dropped MSNLP.
3.3.1.3.14 Octeract

- The GAMS/Octeract link has been rewritten.
  - Temporary files are now removed, unless GAMS option keep is enabled.
  - Octeract is now also available for solvelink=5 and solvelink=6.
  - Octeract options are now documented and Octeract option files can be created via the GAMS Studio options editor.
  - The default for MAX_SOLVER_ITERATIONS is now set to GAMS iterlim.
  - The default for MAX_SOLVER_MEMORY is now set to GAMS workspace.
  - New option nllinear.
- New libraries 4.4.1.

3.3.1.3.15 SCIP

- Added support for logical functions bool_and, bool_eqv, bool_imp, bool_not, bool_or, bool_xor. Only constants, binary variable, and logical functions are allowed as arguments for these operators. The GAMS/SCIP link introduces additional variables and constraints of types and, or, and xor to handle these functions.
- New libraries SCIP 8.0 (504f5f2749).
  - Improved and extended cut selection statistics.

3.3.1.3.16 XPRESS

- New libraries 40.01.01.
  - MIP performance improvements:
    * Improved separation of Mixed Integer Rounding (MIR) cuts in the tree.
    * Improved selection of cutting planes based on orthogonality measures.
    * Some subMIP heuristics at the root node now solve the subMIPs using multiple threads.
    * More advanced MIP presolve operations.
  - Improved crash heuristic for dual simplex.
  - Improved parallel performance of the crossover algorithm.
  - Improved sparse matrix-vector multiplication selection method in dual crossover.
  - New option numericalEmphasis: specify emphasis to place on numerical stability over solving time.
  - Changed default of treeCutSelect to -1.
  - Changed default of treeCutSelect_mirRowAggregation to 1.
- Support mipCleanup also for nonlinear models.

3.3.1.4 Tools

3.3.1.4.1 GAMS IDE

- Added a startup message to announce the drop of the classic GAMS IDE.
3.3.1.4.2 GAMS Studio

- New version 1.11.1.
  - New feature: Allow to open a directory to add all files in that directory to a project.
  - New feature: Added HTML format when copying text to clipboard.
  - GDX Viewer improvements:
    * Saves/restore state (e.g. filters, sorting, ...) when data has to be reloaded from file.
    * Added invert selection for column filter and for the attribute selection.
    * Fixed that auto resize of columns was not triggered for 1-dimensional symbols when the table view has been selected as default view.
  - Added zoom for editors/viewers with proportional fonts (GDX Viewer and Option Editor).
  - Added editor to edit efi files for GAMS Engine.
  - Improved and unified filter line edits for tables and trees to support regular expressions and exact matches.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    * Moved result files from a GAMS Engine run to the working directory.
    * Added link to MIRO output that opens the directory in the file explorer.
    * Improved trimming behavior of search results.
    * Adjusted wording in search results table header.
    * Fixed focus and auto-fill when re-triggering search dialog while it is already open.
    * Fixed "All Files" filter not showing files without extension on Linux and macOS.
    * Fixed error in restoring selection of MIRO deployment dialog.

3.3.1.4.3 GDX2ACCESS

- We will remove the tool gdx2access in a future GAMS release. GAMS Connect agent PandasSQLWriter should be used instead.

3.3.1.4.4 GDX2SQLITE

- We will remove the tool gdx2sqlite together with the tool sqlite3 in a future GAMS release. GAMS Connect agent PandasSQLWriter should be used instead.

3.3.1.4.5 GDXVIEWER

- For the interactive mode added a startup message to announce the drop of the GDXVIEWER.

3.3.1.4.6 MDB2GMS

- We will remove the tool mdb2gms in a future GAMS release. GAMS Connect agent PandasSQLReader should be used instead.
3.3.1.4.7 MPS2GMS

- Fixed that non-continuous variables in cones (CSECTION) were missing in =c= equations in written GAMS file.
- Polished written GAMS file, in particular eliminate use of $batinclude$.
- Quadratic cones (CSECTION) are now reformulated into quadratic equations. New option CEQUATIONS can be used to enable previous behavior of writing =c= equations.
- Input files that were compressed with gzip (.gz) can now be handled.
- Added support for input in LP format.
- Updated MPS reader to HiGHS 892691737.

3.3.1.4.8 SQL2GMS

- We will remove the tool sql2gms in a future GAMS release. GAMS Connect agent PandasSQLReader should be used instead.

3.3.1.5 APIs

3.3.1.5.1 GAMS Transfer Python

- Modified behavior of registering UELs: Previously, GAMS Transfer generated UELs from data in record order (i.e., all UELs from the first data row are registered, then all UELs from the next row are registered, etc.). GAMS Transfer now registers UELs columnwise (i.e., all UELs from the first domain column are registered, then all UELs from the next domain column are registered, etc.). This behavior change will impact symbols written over "*" or with relaxed domains.
- New Symbol methods hasDuplicateRecords, countDuplicateRecords, findDuplicateRecords, and dropDuplicateRecords to help debug and resolve errors with duplicate records.
- New Container methods hasDuplicateRecords, countDuplicateRecords, and dropDuplicateRecords to help find symbols that contain duplicate records that cause the Symbol to be invalid.
- New Symbol methods hasDomainViolations, countDomainViolations, and dropDomainViolations to help debug and resolve errors with domain violations.
- New Container methods hasDomainViolations, countDomainViolations, and dropDomainViolations to help debug and resolve errors with domain violations.
- Fixed that the column headings of the records DataFrame were not updated when a domain set/alias was renamed or the .domain attribute of a symbol was updated directly.
- Fixed read of equation with unknown subtype (recast as =E/=eq).

3.3.1.5.2 GAMS Transfer R

- New API GAMS Transfer R to exchange data between GAMS and R.

3.3.1.5.3 Python

- As announced, dropped support for Python 3.6.
3.3.1.6 Model Libraries

3.3.1.6.1 GAMS Data Library

- Added `connect04.gms` : Simple Connect Example for SQL.

3.3.1.6.2 GAMS Test Library

- Added `load15.gms` : Test consistent compile time error checking when loading symbols.
- Added `gurobi07.gms` : GUROBI test suite - nonlinear constraints to general constraints.
- Added `load16.gms` : Test implicit dimension setting when loading a symbol.
- Added `gamsxcppmex.gms` : Test calling and validating new C++ compiler against legacy Delphi compiler.
- Added `gdxdiff2.gms` : Test GDXDIFFs fdonly option.
- Added `capdsqr.gms` : Test Connect agent PandasSqlReader.
- Added `capdsqwl.gms` : Test Connect agents PandasSqlWriter.
- Added `cadomainw.gms` : Test Connect agent DomainWriter.

3.4 39 Distribution

3.4.1 39.1.0 Major release (May 03, 2022)

3.4.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Wolfgang Britz, Rob Davies, Torbjoern Jansson, Bruce McCarl, and Yihe Zhuo.

3.4.1.2 GAMS System

3.4.1.2.1 GAMS

- Added basic model statistics (range of absolute non-zero values for the bounds, right hand sides and matrix coefficients) to the output file, and log.
- Added model attributes to access the values of the basic model statistics also programmatically: `rngBndMax`, `rngBndMin`, `rngMatMax`, `rngMatMin`, `rngRhsMax`, `rngRhsMin`.
- Added new command line parameter `multi` to initialize the state of `$on/offMuli` and `$on/offMultiR`.
- Extended the syntax of `$[on][off]Echo[V]$, `$[on][off]Put[V]$, `$[on][off]EmbeddedCode[V]`, and `$[end][pause]EmbeddedCode[V]` with an optional suffix to allow nested use. Note that this might trigger an error for code that was OK before. In the following example, the suffix at `$onEcho` was ignored in the past. Now, this will result in an error, since there is no `$offEcho` with matching suffix:
In embedded code Python, the Python interpreter stays alive even when the embedded code section ends with \texttt{$\text{offEmbeddedCode}$} and \texttt{endEmbeddedCode}. So one has access to Python objects across embedded Python code sections. Nevertheless, the \texttt{gams} object gets newly initialized (with the exception of \texttt{continueEmbeddedCode}). Hence old references to \texttt{gams} or derived objects, e.g. \texttt{sym = gams.get("sym")} still exist in Python but the connected external resources are no longer available. Accessing these objects results in a crash as demonstrated in the following example:

\begin{verbatim}
Set s1 / i1*i3 /, s2 / j1*j3 /;
parameter p1(s1, s2);
$onEmbeddedCode Python:
s1 = gams.get('s1')
s2 = gams.get('s2')
$offEmbeddedCode

$onEmbeddedCode Python:
p1 = [(i, j, 3.14) for i,j in zip(s1, s2) ]
gams.set('p1', p1)
$offEmbeddedCode
\end{verbatim}

In the second embedded Python code section the object references \texttt{s1} and \texttt{s2} use unavailable external resources, and the code in the section will crash GAMS. There is no need to use \texttt{gams} or derived objects from previous embedded code sections, since one can always use the current \texttt{gams} object. For legacy code that makes use of this unintended feature, we allow the use of the deprecated (and hidden) command line option \texttt{freeEmbeddedPython=0} for some time to get the old behavior. If you find yourself in need to set this option, you need to rework your embedded Python code section to prevent use of previous \texttt{gams} and derived objects. The option \texttt{freeEmbeddedPython} will (as announced) eventually be removed.

- Added new command line parameter and option \texttt{ECImplicitLoad}, as well as related dollar control options \texttt{$\text{on/offECImplicitLoad}$}.
- Added new command line parameters \texttt{ConnectIn} and \texttt{ConnectOut} to instruct GAMS to pass the Connect YAML files provided by these option to the Connect framework at the beginning of a GAMS job (\texttt{ConnectIn}) and at the end (\texttt{ConnectOut}) of the GAMS job.
- Added new \texttt{put utility} command \texttt{ECArguments} to extend the embedded code \texttt{arguments} with a dynamic text.
- Added explicit range checking for some \texttt{Put File Attributes}. In the past, unexpected values for some attributes (e.g. \texttt{file.nr = 8;}) could have led to unwanted output. Now, unacceptable values get projected to the nearest acceptable value.
- Updated use of decimals (\texttt{file.nd}) value in floating-point put output. Previously, the decimals value was treated as a specification, with putfile error messages resulting if the output contained fewer decimals. We treat the decimals value as a limit now: it’s not an error if fewer digits are shown because of the width limit or for other reasons.
- Removed the GDX file \texttt{gmspfmap.gdx} that contained some partial mapping of files to components from the system directory.
- There was a hidden option \texttt{work}. This cannot be used anymore. The command line parameter or model attribute \texttt{workSpace} should be used instead.
- Allow \texttt{holdFixed=1} for EMP, but not EMPSP models. The partly reverts the change of \texttt{holdFixed} for EMP models from GAMS 38.
- Fixed a bug where the command line parameters \texttt{holdFixed}, \texttt{nodLim}, \texttt{workFactor} and \texttt{workSpace} took precedence over the related model attributes.
• Fixed an error with constant evaluation in data statements for scalar symbols in a parameter statement.

• Fixed output of very large negative numbers (e.g. -DBL_MAX) to trace files.

• Fixed minor rounding and display issues with double-precision numbers. This is relevant to values displayed in the listing file, put files, and less commonly in the GAMS log. Ties are now resolved using the round-to-even rule instead of round-towards-zero. This new tie-breaking rule is consistent with the display routines in C++, Python, Julia, and R. For example:

  – with option decimals=5, scalar p / 9999998.4/ was displayed as 1.00000E+7 – now as 9.999998E+6
  – with option dispwidth=9, decimals=5, scalar p / -8.123441235/ was displayed as -8.123E+1 – now as -8.1234412
  – very small values (1e-250 or less) were displayed as 0 but are now displayed using E-format in the usual way
  – displaying 1.5 or 2.5 with zero decimals both now show 2, while previously 1.5 would display as 1
  – similarly, 1.75 with decimals=1 now displays as 1.8 instead of 1.7 as it did previously

3.4.1.2.2 Connect

• We have added the Connect framework to the distribution. Connect gives unified and platform independent access to data exchange with different formats (e.g., CSV and Excel). The instructions how to access the various data sources are given in YAML syntax. Connect is available as a standalone command line utility (gamsconnect), via the GAMS command line parameters ConnectIn and ConnectOut, and via embedded Connect code. The Connect framework is currently in beta status.

3.4.1.2.3 Embedded Code

• Allow implicit loading of symbols in embedded code. In the past the embedded code had to "set" a symbol and the symbol has to be explicitly imported on the $off/end/pauseEmbeddedCode line. This is no longer necessary. By default, a symbol that has been "set" by the embedded code will be imported into GAMS. The behavior with respect to this implicit loading is controlled by the command line parameter ECImplicitLoad and the dollar control options $on/offECImplicitLoad.

3.4.1.2.4 Embedded Connect Code Facility

• We added the new embedded code engine Connect. The embedded Connect code gives access to the Connect framework. The YAML formatted "code" can be utilized to import and export data with GAMS during compile and execution time.

3.4.1.2.5 Embedded Python Code Facility

• The Python version of GMSPython has been updated to 3.8.13.
• Added the packages cerberus and pyyaml-include to GMSPython.
• Removed decoration (---) from log messages printed using gams.printLog() and fixed a bug that prevented long messages from being displayed correctly.
3.4.1.2.6 Extrinsic Function libraries

- As announced the extrinsic function library parcclib has been dropped.
- The mutex library mtxclib, hidden in the system since 24.6, has now been documented and modified. With this version a mutex first has to be created via the function Create before it can be used (e.g. locked).

3.4.1.2.7 Windows Installer

- The installer no longer creates a Desktop icon for the GAMS IDE by default.
- Fixed bug that prevented command line parameter /desktopIcons=no from having an effect.

3.4.1.3 Solvers

3.4.1.3.1 CONOPT

- New libraries 3.17M.
  - Fixed issue with incorrect infeasibility reporting for infeasible CNS models or models with infinite post-triangular variables.
- New libraries 4.27.
  - Improved preprocessor on models with very small coefficients and models with constant nonlinear Jacobian elements.
  - Improved detection of post-triangular parts in CNS models.
  - Improved speed of analyzing the numerical stability and density of definitional constraints.
  - Improved numerical stability of creating and testing the initial basis.
  - Added options:
    * Flg_TraceCNS: Flag for tracing a CNS solution.
    * Trace_MinStep: Minimum step between reinversions when using TraceCNS.

3.4.1.3.2 COPT

- New libraries 4.0.5.
  - Improved performance for LP and MIP solving.
  - Added support for convex quadratic equations in continuous problems (model types QCP and RMIQCP). Currently, marginals are not provided if quadratic constraints are present.
  - Added option to find an Irreducible Inconsistent Subsystem (IIS) for an infeasible LP or MIP. New options iis to enable the IIS finder and IISMethod to specify IIS finder method.
  - New parameter BarOrder to specify ordering algorithm in Barrier solver.
  - For GAMS/COPT-Link users: If no COPT license is available, COPT can still be used for non-commercial use and LPs with at most 10000 variables and 10000 constraints and for MIPs/QCPs with at most 2000 variables and 2000 constraints.
3.4.1.3.3 CPLEX

- New libraries 22.1.0.
  - New options:
    * lowerobjstop: in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value.
    * upperobjstop: in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value.
  - Removed CPLEX Remote Object Server and Distributed MIP.
  - Removed options `computeserver`, `polishtime`, `rampuptimelimit`, `rampupduration` and `rampuptimelimit`.
- New option `exactkappa`: Report exact condition number in quality report. Note: Previously, the exact condition number has always been reported. Now, default is to not do it due to possibly high computation times.
- Add previously hidden options:
  - `iafile`: secondary option file to be read in interactive mode triggered by `iatriggerfile`.
  - `iatriggerfile`: file that triggers the reading of a secondary option file in interactive mode.
  - `iatriggerertime`: time interval in seconds the link looks for the trigger file in interactive mode.
  - `ltol`: basis identification primal tolerance.
  - `mtol`: basis identification dual tolerance.
  - `readparams`: read CPLEX parameter file.

3.4.1.3.4 DE

- Introduced an experimental and hidden option `pipsStages` to annotate the resulting deterministic equivalent with PIPS-IPM++ conforming block information.

3.4.1.3.5 EMPSP

- For EMPSP models feasible empty rows will no longer be sent to the solver.

3.4.1.3.6 GLOMIQO

- We plan to drop GLOMIQO with the next major release. The ANTIGONE solver (a superset of GLOMIQO) remains in the system.

3.4.1.3.7 GUSS/ScenarioSolver

- GUSS/ScenarioSolver allows now to manually scale model instances via the `.scale` variable and equation suffix together with model attribute `scaleOpt=1`.

3.4.1.3.8 Ipopt

3.4.1.3.9 KNITRO

- Added support for models with more than $2^{31}$ entries in the coefficients matrix.
- Changed handling of initial levels and marginals, see initvalues0 and initvalues.
- Only scale the model instance if scaleOpt is set to 1.
- Added options:
  - initvalues0: enable use of initial guess for levels and marginals (first solve).
  - initvalues: enable use of initial guess for levels and marginals (subsequent solve).

3.4.1.3.10 LGO

- We plan to drop LGO with the next major release.

3.4.1.3.11 Lindo/LindoGlobal


3.4.1.3.12 MOSEK

- New libraries 9.3.20.
- For GAMS/MOSEK and GAMS/OsiMosek, a separate MOSEK license is now required also for academic GAMS licenses that include a GAMS/MOSEK link license but no full GAMS/MOSEK license.

3.4.1.3.13 MSNLP

- We plan to drop MSNLP with the next major release.

3.4.1.3.14 Octeract

- New libraries 4.3.1.
  - Fixed that GAMS options were ignored when a solver options file was specified.

3.4.1.3.15 ODHCPLEX

- New libraries 6.11.

3.4.1.3.16 SCIP Optimization Suite

- New libraries PaPILO 2.0 (60ab076).
- New libraries SoPlex 6.0 (f084e555).
- New libraries SCIP 8.0 (899dc92dda).
3.4.1.3.17 XA

- We plan to drop XA with the next major release.

3.4.1.3.18 XPRESS

- New libraries 39.01.06.

3.4.1.4 Tools

3.4.1.4.1 GAMS Studio

- New version 1.10.1.
  - New feature: Added pin view: The main edit area can be split vertically or horizontally to show a second editor.
    - Ctrl + Left Click on a tab creates a vertical split.
    - Ctrl + Shift + Left Click on a tab creates a horizontal split.
  - Added new search scope: Folder (aka Find in Files).
    - This allows users to search a directory on disk without having to load the whole directory in Studio first.
  - Some rework of the "Search and Replace" dialog:
    - Added file exclusion pattern input field and functionality.
    - Added info about number of files searched to search dialog.
    - Changed default position of search dialog to top right.
    - Search now respects open file behavior settings when jumping to results in unopened files.
    - Fixed crash in search when current editor has no file associated.
    - Fixed search not being interruptible when searching many small files.
  - Added GDX Viewer tab to the settings dialog.
  - Added new setting to specify the default symbol view of the GDX Viewer.
  - User model libraries have been moved to the first tab of the Model Library Explorer.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    - Configuration editor now displays gamsconfig.yaml file only.
    - Added search related group to project explorer when files were found that were unknown to Studio.
    - Added automatic deactivation of search selection when leaving "Selection" search scope.
    - Added feedback for empty search term, invalid regex, and invalid path in search dialog.
    - Added completer handling for $offEcho.
    - Changed behavior of completer: Allow all keywords after opening a paired dollar control option (DCO) and showing closing DCOs first.
    - Ensure unique project name and keep log name in sync with project name.
    - Fixed crash in ModelLibrary dialog on Ctrl-rightClick on entry.
    - Fixed white table headers in dark mode.
    - Fixed code completer activation issues.
    - Fixed broken syntax in data statements with division.
### 3.4.1.4.2 MPS2GMS

- The tool has been reimplemented and now uses the readers of the HiGHS solver.
- If the MPS file appears to be in fixed form (row or character names with spaces and at most 8 characters), then parsing in fixed form is attempted.
- GDX and GMS files are now also written if no GDX and GMS filenames are provided explicitly.
- Dropped support for writing an MPS file.
- Dropped support for OBJNAME section.
- Sections SOS and SETS are now assumed to follow the format used by CPLEX, GUROBI, and FICO XPRESS.
- Dropped parameters NAME, N, RHS, RANGES, BOUNDS, MPSOut, Trace.
- Added parameter COLUMNINTVARSAREBINARY to specify how to handle integer variables that appear first in COLUMNS section.
- Quadratic coefficient matrices are no longer written in symmetric form. Zero coefficients (eps) that were written to GDX in certain situations are now omitted.
- See also the MPS2GMS documentation.

### 3.4.1.5 APIs

#### 3.4.1.5.1 GAMS Transfer Matlab

- New version 0.2.0:
  - Improved documentation.
  - Added "equals" method to Container and Symbol classes to compare containers or symbols, respectively.
  - Added "copy" method to Symbol classes to copy symbols to another container.
  - Added support to read symbols directly from Container (not just ConstContainer).

#### 3.4.1.5.2 GAMS Transfer Python

- GAMS Transfer will now accept any mixed-case string representations of special values for undef, na, and eps (i.e., Undef or UnDef are now valid).
- Performance enhancements associated with reading/writing GAMS special values.
- Performance enhancement in .removeSymbols(): isValid check flag is no longer reset for all symbols in the Container if removing a Parameter, Variable or Equation.
- Fixed bug when replacing string special values in a dataframe whose index was not uniformly spaced.
- Fixed bug when testing scalar values with .isUndef.

#### 3.4.1.5.3 GMD

- Special value -0.0 for EPS is now properly recognized.
- gmdCopySymbol allows to copy parameter, variable, and equation symbols into a set symbol.
- Added gmdGetUserSpecialValues to access special values GMD currently accepts.
3.4.1.5.4 GMO

- Added properties `gmoHessNZ64`, `gmoHessMaxNZ64`, and `gmoHessLagNZ64` and functions `gmoHessStruct64` and `gmoHessValue64` to get Hessian element counts as 64-bit integer.

- Added properties `gmoQN64` and `gmoQNL64`: all properties returning 32-bit Jacobian nonzero counts now have 64-bit counterparts.

- Fixed a memory leak in the Hessian evaluation: the leak occurred during speculative Hessian evaluations, i.e., evaluations with a tight memory limit intended to see how memory-intensive Hessian evaluation is and if Hessians should be used by the solver.

3.4.1.5.5 High-level APIs

- `GamsModelInstance.solve()` falls back to a warm start in case a hot start fails.

- Added read property for `GAMSSet.SetType` (.NET, Python) and `GAMSSet.getSetType` (Java) to distinguish multi and singleton sets.

- New optional argument `setType=SetType.multi` in `GAMSDatabase.AddSet()` (.NET), `GamsDatabase.add_set()` and `GamsDatabase.add_set_dc()` (Python), and `GAMSDatabase.addSet` (Java) to set type of set.

- Fixed defined enumerated value for options `FDOpt` and `DumpOpt` in Java interface.

3.4.1.6 Model Libraries

3.4.1.6.1 GAMS Model Library

- Dropped `deploy`.

- Dropped `dplytst`.

- Updated `cta.gms`: Controlled Tabular Adjustments.

- Updated `dicegrid.gms`: MIP Decomposition and Parallel Grid Submission - DICE Example.

- Updated `herves.gms`: Herves (Transposable Element) Activity Calculations.

- Updated `qmeanvar.gms`: Financial Optimization: Risk Management using MIQCP.

- Updated `sddp.gms`: Multi-stage Stochastic Water Reservoir Model solved with SDDP.

- Updated `siders.gms`: ERS Data Manipulations with SED.

- Updated `tsp5.gms`: TSP solution with Miller et al subtour elimination.

- Added `asyncincbi.gms`: Asynchronous processing of incumbents reported by GAMS/CPLEX.

- Added `springchain.gms`: Equilibrium of System with Piecewise Linear Springs.

- Added `trussm.gms`: Truss Topology Design with Multiple Loads.
3.4.1.6.2 GAMS Data Library

- Added `gtmvn.gms`: Demonstrate the use of numpy.multivariate_normal on stock return data using Transfer.
- Added `connect01.gms`: Complex Connect Example with Spreadsheets.
- Added `connect02.gms`: Complex Connect Example with CSV Files.
- Added `connect03.gms`: Simple Connect Example with CSV Files.

3.4.1.6.3 GAMS Test Library

- Updated `embpy04.gms`: Test continuation of embedded code blocks to remove tests related to freeEmbeddedPython.
- Updated `mtxlib.gms`: Test basics of the mutex library to reflect changes in extrinsic function mutex library.
- Dropped `parlib01` and corresponding library `[lib]parcclib64` and source `parsource.zip`.
- Dropped `pfmaptst`.
- Added `embmihws.gms`: ModelInstance hot/warmstart switch.
- Added `convert17.gms`: CONVERT test suite - check stage and scale export in dumpgdx.
- Added `caproject.gms`: Test Connect agent Projection.
- Added `cagamsrw.gms`: Test Connect agent GAMSReader/Writer.
- Added `cagdxrw.gms`: Test Connect agent GDXReader/Writer.
- Added `capdxlsrw.gms`: Test Connect agents PandasExcelReader and PandasExcelWriter.
- Added `carxr.gms`: Test Connect agent RawExcelReader.
- Added `capcode.gms`: Test Connect agent PythonCode.
- Added `cacsww.gms`: Test Connect agent CSVWriter.
- Added `embpy12.gms`: Test implicit/explicit loading in Embedded Code.
- Added `cacsrvr.gms`: Test Connect agent CSVReader.
- Added `capdxlsr.gms`: Test Connect agent PandasExcelReader.
- Added `embpy13.gms`: Test implicit/explicit loading in Embedded Code (execution time).
- Added `connectsub.gms`: Test substitution for Connect.
- Added `nestblock01.gms`: Test the use of nested blocks.
3.4.1.6.4 FIN Library

- Use Connect instead of Windows-only gdxxrw to allow running the models on all platforms.
- Updated CreditImmunization: Factor Immunization model for corporate bonds.
- Updated MeanVar: Mean-variance efficient portfolios.
- Updated MeanVarShort: Mean-variance model allowing short sales.
- Updated PutCall: Put/Call efficient frontier model.
- Updated ReadData: Reads excel files and converts them to gdx format.
- Updated SelectiveHedging: Scenario Optimization for selective hedging.
- Updated ThreeStageSPDA: A three stage stochastic programming model for SPDA.

3.4.1.6.5 PSOPT Library

- Use Connect instead of Windows-only gdxxrw to allow running the models on all platforms.
- Updated DED: Dynamic Economic Load Dispatch.
- Updated DED-PB: Price based Dynamic Economic Load Dispatch.
- Updated DED-wind: Dynamic Economic Load Dispatch considering Wind generation.
- Updated DEDESS: Cost based Dynamic Economic Dispatch integrated with Energy Storage.
- Updated DEDESSwind: Cost based Dynamic Economic Dispatch integrated with Energy Storage and Wind.
- Updated ESSDCOPFwind: DC-OPF integrated with Energy Storage and Wind.
- Updated MultiperiodACOPF24bus: Multi-period AC-OPF for IEEE 24-bus network considering wind and load shedding.
- Updated MultiperiodDCOPF24bus: Multi-period DC-OPF for IEEE 24-bus network considering wind and load shedding.
- Updated PBUC: Price based Unit commitment.
- Updated RampSenDED: Ramp rate sensitivity analysis for Dynamic Economic Load Dispatch.

3.4.1.7 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th></th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COPT 4.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 22.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Solver</td>
<td>x86 64bit MS Windows</td>
<td>x86 64bit Linux</td>
<td>x86 64bit Mac OS X</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUGUROBI 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLNP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCTERACT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QUADMINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 8.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soplex 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 39.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.4.2 39.1.1 Maintenance release (May 18, 2022)

#### 3.4.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz and Arne Schulz.

#### 3.4.2.2 Solvers

##### 3.4.2.2.1 CPLEX

- Fixed `mipkappastats` report in case of relaxed or fixed solve after MIP solve (enabled by `rerun = no` and `solvefinal = yes`, respectively).

##### 3.4.2.2.2 KNITRO

- Fixed incorrect detection as an infeasible model when it actually was feasible (happened if KNITRO terminated due to small relative change in objective).
3.4.2.3 Tools

3.4.2.3.1 GAMSCONNECT

- Fixed problem with gamsconnect command line tool not working properly when environment variables PYTHONPATH or PYTHONHOME were set.

3.4.2.3.2 GAMS Studio

- New version 1.10.2 with some bug fixes, stability improvements, and minor enhancements, e.g.:
  - Made sure the correct file gets the focus on Studio start-up.
  - Fixed crash on closing a tab when project settings are active.
  - Fixed closing behavior of search dialog on Windows when using X button.

3.4.2.3.3 GDXDIFF

- Fixed wrong differences of parameters with option fldOnly.

3.4.2.3.4 MODEL2TEX

- Fixed problem with model2tex not working properly when environment variables PYTHONPATH or PYTHONHOME were set.

3.4.3 39.2.0 Minor release (June 02, 2022)

3.4.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Dominic Lencz.

3.4.3.2 GAMS System

3.4.3.2.1 GAMS

- Fixed potential wrong dump file with implicit sets.

3.4.3.2.2 Connect

- Fixed a problem in GAMSWriter agent that registered unused labels with GAMS.

3.4.3.2.3 Embedded Python Code Facility

- Fixed a problem with ignoring gams.ws.my_eps.
3.4.3.3 Solvers

3.4.3.3.1 CONOPT

- New libraries 4.28.

3.4.3.4 Tools

3.4.3.4.1 GAMS Studio

- New version 1.10.4 with some bug fixes, and minor enhancements, namely:
  - Added support for optional suffixes for $\text{[on][off][Echo][S][V]}$, $\text{[on][off][Put][S][V]}$, $\text{[on][off][EmbeddedCode][S][V]}$, and $\text{[end][pause][EmbeddedCode][S][V]}$.
  - Fixed GAMS Engine not adding files in subdirectories from $\ast .efi$ file.
  - Fixed problem receiving results from Engine when there is an execution error.

3.4.3.5 APIs

3.4.3.5.1 GAMS Transfer Matlab

- New version 0.2.1:
  - Fixed read of variable with unknown subtype (recast as free).
  - Fixed check of variable type on variable creation.
  - Fixed $\text{isEps}$, $\text{isNA}$, and $\text{isUndef}$ of $\text{SpecialValues}$ for sparse matrix input.
  - Fixed default values of external, conic, and boolean equations.

3.4.3.5.2 GAMS Transfer Python

- Fixed read of variable with unknown subtype (recast as free).
- Fixed default values of equation subtypes ($\text{eq}$, $\text{geq}$, $\text{leq}$, $\text{cone}$, $\text{external}$, and $\text{boolean}$).

3.4.3.5.3 GMD

- Fixed a problem with aliases to the universe.

3.4.4 39.2.1 Maintenance release (June 21, 2022)

3.4.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz and Erwin Kalvelagen.
3.4.4.2 GAMS System

3.4.4.2.1 GAMS

- Report correct number in Solve Summary if iteration count exceeds \texttt{maxInt}.
- Added missing entry for empty scalar equations to the tree view of the output file.

3.4.4.2.2 Connect

- Fixed a problem with \texttt{indexSubstitutions} in \texttt{CSVReader} agent where labels in the last index column were not substituted when reading a set without text columns.

3.4.4.3 Solvers

3.4.4.3.1 CPLEX

- Fixed crash after singular basis error.
- Fixed overflow in node and iteration counts.

3.4.4.3.2 SHOT

- New libraries 1.1 (c9bc78df).

3.4.5 39.3.0 Minor release (July 07, 2022)

3.4.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Bruce McCarl and Tom Rutherford.

3.4.5.2 GAMS System

3.4.5.2.1 GAMS

- Fixed a problem that limited the number of characters written to a single line in the log using \texttt{put files} to 255.

3.4.5.3 Solvers

3.4.5.3.1 SCIP Optimization Suite

- New libraries SCIP 8.0 (95d63636d5).
- New libraries PaPILO 2.1.0.
3.4.5.3.2 SHOT

- New libraries 1.1 (033622c6).

3.4.5.4 Tools

3.4.5.4.1 GAMS Studio

- New version 1.10.5 with some bug fixes, and minor enhancements, for example:
  - Fixed Studio ignoring setting to open in current project when opening a file from Explorer.
  - Fixed search not reacting to changes in project file list.
  - Fixed search not always adding new files to correct search group.
  - Fixed potential crash when closing Studio while search dialog is open.
  - Fixed opening of empty editor when using search results to open a deleted file.
  - Added automatic update of results list when still open from last search.

3.4.5.5 APIs

3.4.5.5.1 GMO

- Fixed a crash caused by memory corruption in the Hessian evaluation: the crash occurred while cleaning up after speculative Hessian evaluations that exceed a user-specified memory limit.

3.5 38 Distribution

3.5.1 38.1.0 Major release (January 31, 2022)

3.5.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Albert Brouwer, Arne Drud, Daniel Jungen, Scott McDonald, Tom Rutherford, Seyed Amin Sedgh, and Wenjin Zhou.

3.5.1.2 Platforms

- As announced, dropped support for Mac OS X 10.14 (Mojave).
- Added support for Mac OS X 12 (Monterey).
3.5.1.3 GAMS System

3.5.1.3.1 GAMS

- Added new command line parameter `dumpOptGDX` to load the data needed in `dumpOpt` files from GDX instead of data statements.
- Added new command line parameter `ReferenceLineNo` to control the line numbers written to a reference file.
- We plan to drop the command line parameter `freeEmbeddedPython` with the next major release.
- Extended the Solve Summary written with asyncSolLst=1 to include information about controlling loop indices during job submission, name of the solver called, source line number of the solve statement, resource and iteration limit.
- Make handling of execution errors during model generation (e.g. division by zero in an equation) more consistent across different settings for the `solveLink` option. In the past, the error count was set to 0 automatically when returning from a solve with `solveLink=0` only (which is equivalent to setting `execError=0` after the solve statement). This is not done anymore.
- Write additional information about execution errors during model generation (e.g. division by zero error) to the lst file, e.g., the equation name and index causing the problem.
- Improve recognition of `/` as path delimiter on Windows.
- Pause automated time dependent log update while executing embedded code sections.
- Check for certain byte order marks (BOMs) in GAMS input files. Files with UTF8 BOM can be processed now, other BOMs generate a compilation error.
- Added support for (mixed-integer) quadratic models whose Q matrices contain more than $2^{31} = 2,147,483,648$ nonzeros in their upper triangle. At this time, only the solver links for CPLEX and ODHCPLEX are capable of handling such large model instances.
- Changed system suffix `isAlfaBeta` to `isAlphaBeta`.
- Fixed a problem for EMP models with `holdFixed` model attribute.
- Fixed a problem where IDCGenerateGDXInput, IDCGenerateGDXOutput, IDCGDXOutput, and IDCGenerateGDX did not write a GDX file if the execution was stopped with `abort[.noError]`.

3.5.1.3.2 Extended Mathematical Programming (EMP)

- Stochastic EMP: Changed the `stage` behavior so that it creates an explicit error when a stage gets assigned twice for the symbol. In the past, this caused erratic behavior.

3.5.1.3.3 Embedded Python Code Facility

- The Python version of GMSPython has been updated to 3.8.12.

3.5.1.3.4 Extrinsic Function libraries

- With the next distribution we will drop the extrinsic function library `parcclib`. 
3.5.1.4 Solvers

3.5.1.4.1 BARON

- New libraries 22.1.16.
  - new presolve implementation with novel presolve techniques for linear and nonlinear optimization problems improves performance for many problem classes
  - more classes of SOCPs are now recognized
  - new lift-and-project implementation that extends the previous RLT implementations to generic NLPs and MINLPs and leads to substantial reduction of duality gaps for difficult problems
  - improved tree management, including restarts of branch-and-bound search
  - improved branching routines for the identification of most important branching variables
  - dynamic LP algorithm selection strategy improves robustness for numerically difficult problems by automatically choosing between CPLEX (if licensed) and CLP/CBC and possibly switching during the search, see also option LPSol
  - improved control on the effect of floating point round-off errors
  - faster hashing routines
  - the embedded FilterSD algorithm is now off by default
  - better integration with GAMS NLP subsolvers
  - improvements to CBC and IPOPT interfaces
  - update the Xpress interface to work with FICO XPRESS 39
- Fixed reporting of proven irreducible infeasible sets ("IIS" instead of "(I)IS").

3.5.1.4.2 CONOPT

- New libraries 4.25.
  - General improvements in the construction of the initial basis and in the startup of the optimization process, especially for very large models.
  - Many small improvements and corrections of errors found by users.

3.5.1.4.3 CONVERT

- The symbol $\mathbf{w}$ in DumpGDX is now stored sparse.
- Changed option default of GDXHessian and GDXQuadratic to 0.

3.5.1.4.4 COPT

- For GAMS demo and community licenses, model size limitations as for other solvers apply. For an unlimited GAMS/COPT or GAMS/COPT-Link license, contact sales@gams.com.

3.5.1.4.5 CPLEX

- Fix appending userjobid to userincbicall, usergdxin, and usergdxnameinc in BCH calls.
- Fixed computing dual multiplier at cone top (QCP models).
3.5.1.4.6 DICOPT

- Fixed that non-default stage information was setup in sub-MIPs when prioropt=0.

3.5.1.4.7 KNITRO

- New libraries 13.0.
  - Introduces an updated, parallel, branch-and-bound solver for mixed-integer optimization. This updated solver is able to exploit parallelism and uses improved heuristics to solve mixed-integer problems much faster than before.
  - Offers new initial point strategies for non-convex quadratic programs (QPs) and quadratically constrained quadratic programs (QCQPs). These new initial point strategies improve the likelihood of finding the global solution, and can be used in conjunction with the Knitro multi-start procedure to search for global or better local solutions for non-convex QP and QCQP models.
  - Offers significant robustness and speed improvements on difficult nonlinear optimization problems.
  - Offers improved performance when using the SQP and MISQP algorithms on larger models.
  - Added options:
    * `ncvx_qcp(init)`: Specifies the initialization strategy used for non-convex QPs and QCQPs.
    * `mip_cutoff`: This value specifies the objective cutoff value for MIP.
    * `mip_heuristic_lns`: Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics.
    * `mip_liftproj`: Specifies rules for adding lift and project cuts.
    * `mip_multistart`: Use to enable MIP multi-start at the branch-and-bound level.
    * `mip_numthreads`: Specify the number of threads to use for MIP branch and bound.
  - Renamed options (old option names still available as synonyms but deprecated):
    * `bar_pencons` to `bar_penaltycons`.
    * `bar_penrule` to `bar_penaltyrule`.
    * `mip_integral_gap_abs` to `mip_opt_gap_abs`.
    * `mip_integral_gap_rel` to `mip_opt_gap_rel`.
    * `par_mnumthreads` to `ms_numthreads`.
    * `par_blasnumthreads` to `blas_numthreads`.
    * `par_lsnumthreads` to `linsolver_numthreads`.
    * `tuner_maxtimecpu` to `tuner_maxtime_cpu`.
    * `tuner_maxtimereal` to `tuner_maxtime_real`.
  - Added option values:
    * `blasoption`: 3.
    * `mip_heuristic_diving`: bits 1 - 4.
  - Changed option defaults:
    * `restarts`: -1 (auto).
    * `mip_strong_candlim`: 128.
- Fixed possible error of incorrect Jacobian for QCP models.

3.5.1.4.8 Lindo/LindoGlobal

- New libraries 13.0.319.
3.5.1.4.9 LocalSolver

- As announced, dropped LocalSolver and LocalSolver70.

3.5.1.4.10 MOSEK

- New libraries 9.3.11.
  - Added support for LPs and MIPs with more than $2^{31}$ entries in the coefficients matrix.

3.5.1.4.11 Octeract

- New libraries 4.1.0.
  - Reduced memory requirement and improved performance.

3.5.1.4.12 ODHCPLEX

- New libraries 6.0.8.
  - Added support of solution pool export via solnpool and solnpoolmerge.

3.5.1.4.13 SCIP Optimization Suite

- New libraries PaPILO 2.0.0.
  - Added dual postsolve functionality for LPs.
  - Immediate application of a presolvers reductions if run sequentially.
  - Improvements in performance and functionality of many presolvers.
  - For more details, see [22] and the PaPILO CHANGELOG.
  - Updated TBB to version 2021.5.0.

- New libraries SoPlex 6.0.0.
  - PaPILO can now be enabled as presolver by setting int:simplifier to the new value 2.
  - SoPlex can now be interrupted by Ctrl+C/SIGINT.
  - Changed default for option int:simplifier from 1 to new value 3. The value 1 now means ”auto”.
  - New option real:simplifier_modifyrowfac.
  - For more details, see [22] and the SoPlex CHANGELOG.

- New libraries SCIP 8.0.0.
  - Symmetry handling:
    - New symmetry handling inequalities based on the Schreier Sims table, which are able to handle symmetries of arbitrary kinds of variables.
    - The symmetry code can now heuristically search for orbitopes that do not completely handle a symmetry component and add certain Schreier Sims cuts to such components.
    - In-tree restarts due to tree size estimation have been made compatible with orbital fixing.
    - Improved performance of upgrades to packing/partitioning orbitopes, symresack cover separation, and propagation for orbisack and symresack.
* Symmetry handling has been extended to detect also symmetries between variables that also appear in nonlinear constraints.

- Cutting planes:
  * New separator **mixing** to generate mixing cuts.
  * New separator **rlt** to compute cuts via the reformulation-linearization technique (RLT).
  * New separator **minor** to compute cuts from 2x2 minors of a violated semidefiniteness constraint that is implied by the extended formulation for bilinear products.
  * New separator **interminor** to compute intersection cuts from 2x2 (not only principle) minors of a violated rank-one constraint that is implied by the extended formulation for bilinear products (currently disabled by default).
  * Separator "aggregation" now uses the objective cutoff row as base row and separates lifted cover cuts based on newer lifting function of Letchford and Souli (2019).
  * Separators "strongcg" and "gomory" now share the same computed basis.

- Primal Heuristics:
  * New construction heuristic **DPS** which additionally needs a user-provided decomposition and splits the problem into several sub-SCIPs according to this decomposition. The sub-SCIPs are solved and updated repeatedly until a feasible solution of the original problem is found.
  * PADM can now reoptimize found solution with original objective function.
  * The RENS neighborhood in the heuristic ALNS now fixes fractional variables if necessary for reaching its target fixing rate.
  * Revised and improved heuristic subNLP, in particular choice of NLP iteration limit and starting condition and reuse of subSCIP.

- The handling of algebraic expressions and nonlinear constraints has been rewritten:
  * Support for trigonometric functions sine and cosine and for the entropy function has been added.
  * Improved recognition of common subexpressions, in particular square and bilinear terms.
  * All type of nonlinear constraints, including quadratic and second-order-cones, are now handled by the constraint handler for nonlinear constraints. The specialized constraint handlers have been removed. However, methods that work on a particular nonlinear structure are now implemented via nonlinear handlers (nlhldr).
  * Added nonlinear handler for quadratic, bilinear, convex, concave, and quotient expressions, second-order cone constraints, and perspective tightening for expressions in semi-continuous variables.
  * As before, an extended formulation for nonlinear constraints is constructed. However, this formulation is now handled implicitly and used to construct the LP relaxation only, while the original formulation is used to check feasibility, propagate variable domains, etc. This difference should drastically improved the likelihood that a solution that satisfies the nonlinear constraints in the presolved problem is also feasible for the original problem.
  * Products in nonlinear constraints are no longer disaggregated for the extended formulation.
  * Improved analysis on which nonlinear variables in nonlinear constraints can be increased or decreased without harming feasibility by taking monotonicity into account.
  * Improved linearization of (sums of) products of binary variables, e.g., clique information is now taken into account.
  * When a variable appears in only one concave less-or-equal constraint, it will be fixed to one of its bounds in more cases now.
  * New branching rule for variables in nonlinear constraints that scores candidates based on constraint violation, variable type, and pseudo-costs simultaneously. Variables that are added for the extended formulation of a nonlinear constraint are no longer branched on by default.
  * New strategies to deal with weak estimators for nonlinear functions and small violations of nonlinear constraints.
  * Improved under/overestimation of multidimensional vertex-polyhedral (e.g., multilinear or concave) functions by use of scaling and keeping the cut-generating LP around.
* Added tables to print statistics of nonlinear constraint handler, nonlinear handlers, and handlers for all supported functions.

- Revised and improved interfaces to Ipopt and CppAD, e.g.:
  * Optimization of taped expressions and sparse Hessian evaluation of CppAD are used now.
  * Added table to print statistics on NLP solvers.

- The interfaces to Gurobi and Mosek are thread-safe now.

- Parameters removed:
  * `gams/resolvenlp`
  * `constraints/abspower/*`, `constraints/bivariate/*`, `constraints/quadratic/*`, `constraints/soc/*`
  * `constraints/nonlinear/cutmaxrange`, `constraints/nonlinear/lifeasheasft`, `constraints/nonlinear/reformulate`, `constraints/nonlinear/sepanlpmintcnt`, `constraints/nonlinear/enfocutsremovable`, `constraints/nonlinear/maxexpansionexponent`, `constraints/nonlinear/upgrade/abspower`, `constraints/nonlinear/upgrade/and`, `constraints/nonlinear/upgrade/bivariate`, `constraints/nonlinear/upgrade/quadratic`
  * `constraints/orbitope/usedynamicprop`
  * `heuristics/multistart/nlpminimpr`
  * `heuristics/subnlp/iteroffset`, `heuristics/subnlp/iterquotient`, `heuristics/subnlp/nlpiterlim`, `heuristics/subnlp/nlptimelim`, `heuristics/subnlp/runalways`, `heuristics/subnlp/minimprove`, `heuristics/subnlp/nlpoptfile`, `heuristics/subnlp/resolverfromscratch`, `heuristics/subnlp/resolverfromscratch`
  * `propagating/symmetry/disabledofrestart`
  * `separating/convexproj/nlptimelim`, `separating/gauge/nlptimelim`, `separating/strongcg/dynamiccuts`, `separating/strongcg/maxronds`, `separating/strongcg/maxrondsroot`, `separating/strongcg/maxsepcuts`, `separating/strongcg/maxsepcutsroot`

- Parameters added:
  * `branching/inference/conflictprio`, `branching/inference/cutoffprio`
  * `cuteselection/hybrid/*`
  * `decomposition/disablereplicators`
  * `expr/log/minzerodistance`, `expr/pow/minzerodistance`
  * `heuristics/alns/nodesquotmin`, `heuristics/alns/initializingrc`, `heuristics/alns/maxcallssamesol`
  * `heuristics/dps/*`
  * `heuristics/padm/reoptimize`
  * `heuristics/subnlp/expectedfeas`, `heuristics/subnlp/feasolstol`, `heuristics/subnlp/iterinit`, `heuristics/subnlp/nimitsolves`, `heuristics/subnlp/nodesfactor`, `heuristics/subnlp/nodesoffset`, `heuristics/subnlp/presolveemphasis`, `heuristics/subnlp/setctol`, `heuristics/subnlp/successrateexp`, `heuristics/subnlp/opttol`,
  * `misc/avoidmemout`
  * `nlhdlr/bilinear/*`, `nlhdlr/concave/*`, `nlhdlr/convex/*`, `nlhdlr/default/*`, `nlhdlr/perspective/*`, `nlhdlr/quadratic/*`, `nlhdlr/quotient/*`, `nlhdlr/soc/*`
  * `nlpi/ipopt/*`
  * `presolving/mlp/probfilename`
  * `propagating/obbt/createincons`,

---

**3.5 38 Distribution**

---
* separating/maxcoefficientfractionfacsrowprepr, separating/filtercutpoolrel
* separating/gomory/genbothgomscg, separating/gomory/trystrongcg
* separating/interminor/
* separating/knapsackcover
* separating/minor/
* separating/mixing/
* separating/rlt/
* table/cons_nonlinear/active, table/cutsel/active, table/expr_hdlr/active, table/nlhdlr/active, table/nlhdlr_bilinear/active, table/nlhdlr_quadratic/active, table/nlp/active
* timing/nlpieval

- Parameters changed:
  * default of constraints/nonlinear/maxproprounds changed from 1 to 10
  * default of display/nexternbranchcands/active changed from 2 to 1 also for nonlinear instances
  * default of display/nfrac/active changed from 2 to 1 also if discrete variables are present
  * default of heuristics/dualval/priority changed from 0 to -10
  * default of heuristics/lface/priority changed from -1104000 to -1104010
  * default of heuristics/mutation/priority changed from -1103000 to -1103010
  * default of heuristics/ulpdiving/priority changed from -1003000 to -1003010
  * default of heuristics/repair/priority changed from 0 to -20
  * default of heuristics/simplerounding/priority from 0 to -30
  * default of heuristics/subnlp/forbidfixings from 1 to 0
  * default of heuristics/subnlp/itermin changed from 300 to 20
  * default of SCIPheuristics/subnlp/priority heuristics/subnlp/priority changed from -2000000 to -2000010
  * default of heuristics/trivialnegation/priority changed from 40000 to 39990
  * default of heuristics/trustregion/priority changed from -1102000 to -1102010
  * default of heuristics/trysol/priority changed from -3000000 to -3000010
  * default of misc/usesymmetry changed from 3 to 7 and range changed from \{0,...,3\} to \{0,...,7\}
  * default of presolving/qpkktref/maxrounds changed from -1 to 0
  * default of propagating/obbt/onlynonconvexvargs changed from 0 to 1
  * default of propagating/symmetry/addsymresacks changed from 1 to 0
  * type of propagating/symmetry/recompute restart changed from boolean to int with range \{0,...,2\}
  * default of separating/strongcg/maxbounddist changed from 1 to 0
  * default of separating/strongcg/priority changed from -2000 to -100000
  * replaced separating/minortho by cutselection/hybrid/minortho, replaced separating/minorthoroot by cutselection/hybrid/minorthoroot, replaced separating/dircutoffdistfacs by cutselection/hybrid/dircutoffdistweight, replaced separating/efficacyfacs by cutselection/hybrid/efficacyweight, replaced separating/intsupportfacs by cutselection/hybrid/intsupportfacs, replaced separating/objparallelfacs by cutselection/hybrid/objparallelweight

- For more details, see [22] and the SCIP release notes.

- Added support for GAMS functions \( \text{sin}, \text{cos}, \text{and entropy} \) for GAMS/SCIP.
- Since SCIP has dropped support for \( \text{min}, \text{and max} \), the GAMS/SCIP link reformulates GAMS functions \( \text{min} \) and \( \text{max} \) by use of \( \text{abs} \) now.
- Removed the feature that ran Ipopt from a returned solution that is recognized by SCIP to be not feasible in the original (non-presolved) nonlinear problem in order to obtain a feasible solution.
3.5.1.4.14 SHOT

- New libraries 1.1 (e225a002).

3.5.1.4.15 XPRESS

- New libraries 39.01.03.
  - Improved MIP performance:
    * The Optimizer now uses an ML module to decide whether to use in-tree cutting. See autoCutting.
    * Improved separation of Mixed Integer Rounding (MIR) cuts.
    * Improved cut activation and deactivation strategies.
    * Improved application of strong branching during the tree search.
    * More information carried through a MIP restart.
  - Improved crossover performance after a Barrier solve through automatic objective perturbation. See barObjPerturb.
  - Improved Barrier performance for large problems on CPUs that support AVX2 (requires setting cpuPlatform to -2).
  - New options:
    * autoCutting: Automatically decide whether to generate cutting planes at local nodes in the tree.
    * barObjPerturb: Defines how the barrier perturbs the objective.
    * ioTimeout: Maximum number of seconds to wait for an I/O operation before it is cancelled.
    * maxStallTime: Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent.
    * preCliqueStrategy: Determines how much effort to spend on clique covers in presolve.
    * siftPresolveOps: Determines the presolve operations for solving the subproblems during the sifting algorithm.
  - Changed option default of xslp.mipDefaultAlgorithm: 1.

- Added option fixoptfile: name of option file which is read just before solving the fixed problem.
- Changed default value of option mipAddCutoff and mipRelCutoff to 0.
- Fixed unnecessary solve of fixed MIP (see mipCleanup) when solving a linear program as MIP.

3.5.1.5 Tools

3.5.1.5.1 BIB2GMS

- As announced, dropped the tool bib2gms.

3.5.1.5.2 CSV2GDX

- Check for certain byte order marks (BOMs) in input files. Files with UTF8 BOM can be processed now, other BOMs generate an explicit error.
3.5.1.5.3 GAMS Studio

- New version 1.9.4.
  - Added Project Options editor.
  - Added possibility to create an empty project.
  - Adjusted loading from a model library to add files to active project working directory if "Open in existing project" is checked.
  - Added selection of user instances on GAMS Engine SaaS.
  - Added selection of namespaces on GAMS Engine Server.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    * Expanded MIRO installation location on Mac OS X ifAppBundle has been selected.
    * Adjusted check if the MIRO data contract is available before the MIRO app is deployed.
    * Changed mouse wheel behavior in tab bars:
      - Mouse Wheel Up/Down: moves active tab
      - Ctrl + Mouse Wheel Up/Down: changes active tab
    * Added touch pad support for tab bars.
    * Added highlight in LST file for current section header of LXI viewer.
    * Changed behavior of search in selection: the search selection is now only set once until the user resets it.
    * Changed "Clear" button to remove search selection on first click when present.
    * Fixed non-overridden recurrence of immediate options on command line parameters.
    * Fixed jump behavior of search when using "This File" and navigating different files.
    * Fixed Find Next broken for results across multiple files.
    * Fixed crash on showing "Project Options" for a project without a runnable gms file.
    * Fixed search not updating cache when document has changed.
    * Fixed Clear Selection button not working as intended, when search scope was set to anything but "Selection".
    * Fixed random crash in Search and Replace dialog.
    * Fixed that checks for file type did not ignore casing (gms, Gms, GMS are all valid).
    * Fixed that the default file filter in search dialog expected files to include a '.' character on non-windows platforms.
    * Fixed file filter in search dialog not being ignored for scopes "This File" and "Selection".
    * Fixed search jump behavior when current file is excluded by file filter.
    * Fixed completer and syntax help activation for user defined GAMS source files.

3.5.1.5.4 GDXDIFF

- Reduce size of output GDX file in case of input files without differences.
- Added new option ignoreOrder.

3.5.1.5.5 GDXMRW

- We mark GDXMRW as deprecated and may remove it in a future release. Please use GAMS Transfer Matlab instead. If you encounter issues with the transition to GAMS Transfer Matlab or if you have any feature requests for GAMS Transfer Matlab, do not hesitate to contact us through support.
3.5.1.6.6 **HEXDUMP**

- As announced, dropped the tool `hexdump`.

3.5.1.6 **APIs**

3.5.1.6.1 **C++ high-level API**

- Removed conversion of mapped network drives to UNC paths.
- Fixed addition of GAMS system directory to `PATH` on Windows.

3.5.1.6.2 **GAMS Transfer Matlab**

- New read-only `ConstContainer` (high reading performance).
- `Container` now supports to read from a `ConstContainer` in addition to a GDX file.
- New argument types in `Container::listVariables` and `Container::listEquations` to filter by variable or equation type, respectively.
- `Container::listSets` does not include aliases in the returned list anymore. However, it is still possible to include aliases in `Container::describeSets`.
- New argument `records` in `Container::read` to toggle reading records in addition to symbol meta data.

3.5.1.6.3 **GAMS Transfer Python**

- GAMS Transfer now works with Embedded Python Code.
- New read-only `Container` type called `ConstContainer` (high reading performance).
- New argument types for methods `listVariables()` and `listEquations()` to filter by variable or equation type, respectively.
- The method `listSets()` does not include aliases in the returned list anymore.
- The `Container.read()` argument `values` has been renamed to `records`.
- Fixed bug in `SpecialValues.isNegInf()` that did not properly detect scalar `-inf`.
- Fixed error handling when invalid records are set directly with `.records`.

3.5.1.6.4 **GDX**

- Internal adjustment to detection of special values when writing to GDX. The updated GDX behavior will allow better differentiation between distinct double-precision values that the user wants mapped to distinct special values. For example, -0.0 is no longer confused with +0.0, and different IEEE NaN values can map to different GAMS special values.
3.5.1.6.5 GMO

- Added routines `gmoGetRowQNZOne64`, `gmoMaxQNZ64`, and `gmoObjQMatNZ64` to get element counts as 64-bit integer.
- Adjust routines `gmoGetRowQNZOne`, `gmoMaxQNZ`, and `gmoObjQMatNZ` to return -1 if the element count is too large for a 32-bit integer.
- Added routines `gmoLNZEx` and `gmoLNZEx64` to get the exact count of linear nonzeros in the Jacobian matrix, and adjusted the description of `gmoLNZ` and `gmoLNZ64` to make clear that these are legacy routines returning an overestimate of this count, especially when `gmoUseQ` is true.
- Fix gradient intervals for functions involving `power(x,i)` for `i >= 4` and even.

3.5.1.6.6 High-level APIs

- The `GAMSDatabase` GDX export method now writes `regular` instead of just `relaxed` domain information if available.

3.5.1.6.7 PAL

- Added function `palIsAlpha`. `palIsAlfa` has been deprecated.

3.5.1.6.8 Python

- Added support for Python 3.10.
- We plan to drop support for Python 3.6 in a future GAMS release.

3.5.1.7 Model Libraries

3.5.1.7.1 GAMS Data Library  New model:

- `GMSPythonLib.gms` : `GMSPYTHONLIB` compatibility check (142)

3.5.1.7.2 GAMS Test Library  New models:

- `dumpopt2.gms` : Test `dumpOpt` with `dumpOptGDX`
- `encoding01.gms` : Test processing of source files with BOM
### 3.5.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAEC 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COPT 3.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 20.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINLOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLPR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCTERACT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QUADMINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 8.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 39.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.5.2 38.2.0 Minor release (February 17, 2022)

#### 3.5.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Gwendal Nivanen.
3.5.2.2 GAMS System

3.5.2.2.1 GAMS

- Fixed an incorrect evaluation of intrinsic functions (and their derivatives) having a variable number of arguments (e.g. $edist$ or $poly$). This error did not occur in all cases: it depended on a missing or incorrect initialization in the calling environment.
- Fixed a problem in the extrinsic function declaration which required <$\text{FuncName}$> to be case sensitive.

3.5.2.3 Solvers

3.5.2.3.1 BARON

- New libraries 22.2.3.

3.5.2.3.2 CONOPT

  - Removed option "Flg_Crash_Basis".
  - Changed default of option $\text{Lim_Start_Degen}$ from 10 to 100.

3.5.2.3.3 CONVERT

- Fix missing semi-colon at end of comment in lingo output.
- Fix unary minus in lingo output.

3.5.2.3.4 COPT

- Added option $\text{readparams}$ to instruct COPT to read an additional parameters file.

3.5.2.3.5 CPLEX

- Fix solution pool export: Previously the solution pool was only exported if CPLEX terminated successfully with a solution. Now, the solution pool GDX is exported independent of the CPLEX termination status, but may contain no solution. $\text{solnpoolpoprepeat}$ and $\text{solnpoolpopdel}$ still don't have any effect if CPLEX terminated unsuccessfully.

3.5.2.3.6 GAMSCHK

- Add a missing initialization for the case where no subsolver is called. Without this initialization, incorrect output would occur.
3.5.2.3.7 Ipopt

- New libraries 3.14.5.
  - Fixed that marginals were always zero if Ipopt stopped due to reaching a timelimit.

3.5.2.3.8 XPRESS

- New libraries 39.01.04.

3.5.2.4 APIs

3.5.2.4.1 High-level APIs

- Fixed a problem with `GAMSModelInstance.Instantiate` complaining about a missing dictionary file. This happened when a (default) solver for the model type of the instance is active which does not use a dictionary.

3.5.2.4.2 GAMS Transfer Python

- Added new property `shape` to `Parameter`, `Variable`, and `Equation` to report the corresponding matrix shape if `toDense` or `toSparseCoo` are called.
- Fixed bug with array generation (1D symbols): `toDense` will now generate a `(n,)` array instead of `(1,n)”; harmonized this behavior with `toSparseCoo`.
- Fixed behavior: when `toDense` is called on a scalar symbol an array of shape `(,)` will be returned instead of a `float`.
- Fixed behavior: when `toSparseCoo` is called on a scalar symbol a sparse `coo_matrix` of shape `(1,1)` will be returned instead of a `float`.
- Better handling of numeric domain information within `setRecords` methods: numeric entries are now converted to `str` and generate categories from the `str` equivalents.

3.5.2.4.3 GMO

- The default for the property `gmoHessInclQRows` was accidentally changed to `false` with GAMS 35.1.0. With this release, it has been changed back to its original setting of `true`. That is, also if `useQ` is set, the Hessian will include information on quadratic equations by default again.

3.5.3 38.2.1 Maintenance release (February 19, 2022)

3.5.3.1 Solvers

3.5.3.1.1 CONOPT

- Fixed warning about unknown option name “FLG.CRASH.BASIS” (introduced with GAMS 38.2.0).
3.5.3.1.2 Convert

- Fixed error when exporting CNS models with ampl, jump, lingo and pyomo.

3.5.3.1.3 Octeract

- Avoid that an ObjEst of $\pm 1.797693134E308$ is reported as 1797693134 in GAMS tracefile.

3.5.4 38.3.0 Minor release (April 05, 2022)

3.5.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Daniel Dias, Antti Lehtila, Scott MacDonald, Evangelos Panos, Richard Waltz and Eric Williams.

3.5.4.2 GAMS System

3.5.4.2.1 GAMS

- Fixed wrong numbers about infeasibilities (e.g., in the report summary) when bounds with value EPS are involved.
- Fixed potential unwanted domain violation error when implicit set definition is used while $\$onMulti is active to allow a redefinition of symbol data.
- Fixed potential error when loading data filtered from embedded code.

3.5.4.3 Solvers

3.5.4.3.1 BARON

- New libraries 22.3.21.

3.5.4.3.2 CONVERT

- Fixed export of variable attribute stage for discrete variables when using option DumpGDX.
- Fixed that coefficients in QCMATRIX section were wrong. They had only half its value.
- Fixed missing asterisk for bilinear terms in CplexLP format.

3.5.4.3.3 COPT

- Fixed that indicators from solver options file were ignored.
3.5.4.3.4 CPLEX

- Fixed bug with registering an advanced basis.
- Fixed CPLEX solution status `CPX_STATNUMBEST`. This status is mapped to GAMS model status `feasible`.
- Fixed interactive mode (when no trigger file is set).

3.5.4.3.5 GUROBI

- New libraries 9.5.1.

3.5.4.3.6 IPOPT

- Fixed that names for vector entries corresponding to inequality equations (as shown with `print_level ≥ 8`) could be incorrect due to a wrong mapping of indices.

3.5.4.3.7 KNITRO

- Fixed unnecessary calls to Knitro evaluation callbacks for linear or quadratic constraints.

3.5.4.3.8 Mosek

- New libraries 9.3.18.

3.5.4.3.9 Octeract

- New libraries 4.2.0.
  - Improved performance, in particular for problems with quadratic structures.
  - CPLEX is now automatically chosen as subsolver, if licensed.

3.5.4.3.10 SCIP Optimization Suite

- New libraries SCIP 8.0 (189beea30).
  - Clique lifting for setpartitioning/packing/covering constraints is now only disabled if it has been applied.
- New libraries PaPILO 2.0 (g166dea4).

3.5.4.3.11 XPRESS

- New libraries 39.01.05.
- Changed option default of `xslp_iterLimit`: -1 (auto).
3.5.4.4 Tools

3.5.4.4.1 MODEL2TEX

- Fixed a crash that occurred when `SameAs` was used in equations.

3.5.4.4.2 MPS2GMS

- Fixed that variables in SOS of type 2 were written as SOS of type 1 to GDX file.
- Fixed that upper bounds of 100 for (semi)integer variables were lost.
- Fixed that Σ-rows that did not specify objective functions were not omitted in parameter p (equation right-hand sides).
- Fixed that nonzero coefficients with absolute value at most 1e-20 were lost.
- Fixed that C-rows had arbitrary right-hand side value in GDX file (though this value was not accessed from the GAMS model).
- Fixed that an objective constant (nonzero RHS for objective row in MPS file) was lost.
- Fixed that off-diagonal elements in QUADOBJ and QSECTION sections were not handled correctly.

3.5.4.4.3 GAMS Studio

- New version 1.9.6 with various bug fixes, stability improvements, and minor enhancements, e.g.:
  - Added Page-up/-down keys to extend block edit selection.
  - Added separator line on Windows in horizontal headers of tables and tree views.
  - Remote jobs now can be kept active on Studio exit.
  - Fixed crash in GDX Viewer related to reading a file containing negative UELs.
  - Fixed empty log file when using `logOption=2`.
  - Fixed search not properly counting search results across multiple files.
  - Fixed "Replace All" preview wrongfully including read-only files.
  - Fixed "Search" not always jumping to a result outside current file.
  - Fixed NEOS freezing when the GMS file is not located in the working directory.

3.5.4.5 APIs

3.5.4.5.1 GAMS Transfer Python

- Fixed memory leak associated with `gams2numpy`. 
### 3.6 37 Distribution

#### 3.6.1 37.1.0 Major release (November 11, 2021)

**Acknowledgments**

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Boyan Atanasov, Edna Johanna Molina Bacca, Abhijit Bora, Wolfgang Britz, Bruno Charlier, Hancheng Dai, Rob Davies, Ricardo M. Pinto de Lima, Bruce McCarl, Scott McDonald, Guillaume Nimal, Gwendal Nivanen, Zack Pecenak, Tom Rutherford, Nick Sahinidis, Shima Sasanpour, and Denys Yemshanov.

#### Platforms

- We will drop support for Mac OS X 10.14 (Mojave) with the next major release.

#### GAMS System

**GAMS**

- Added new intrinsic function logit.
- Added new conditional expression onState to check the state of certain dollar control options. An example can be seen here.
- Added command line parameter and option EpsToZero as well as dollar control options $onEpsToZero and $offEpsToZero to allow writing EPS values as zero to GDX.
- Made command line parameters holdFixedAsync, sys15, sys16, sys17, sys18, and sys19 available in option statements as well.
- Avoid warnings if $abort or $call.checkErrorLevel are used within a $ifThen ... $else statement.
- Allow space in limited domain declaration like this: Model transport / all, x (ij) /;
- In the past, GAMS threw an execution error, when execute unload failed because of an invalid file name given. Now, it tries to write to a file with a generic file name first (like it was always done with put files as well).
- Added a new command line parameter and option maxGenericFiles to specify the number of tries to write to a generic file as mentioned above.
- Extended the influence of $offInclude to suppress the include file summary and the creation of an expanded include file as well.
- Added support for (mixed-integer) linear models that generate more than $2^{31} = 2,147,483,648$ nonzeros. At this time, only the solver links for CPLEX, GUROBI, and XPRESS are capable of handling such large model instances. The number of rows and columns is still limited to $2^{31}$.
- Removed a limitation for the model attribute handle that did not allow to set a value which is anything but a Grid handle (specifically, a multi threading handle was rejected before).
- Throw an explicit execution error, if a solver that comes without executable gets called with the Grid Facility (at the moment, this affects DECIS only).
- Fixed calling a solver with the Grid Facility that is not located in the GAMS system directory.
- Fixed alignment of `dmpSym` and `dmpUserSym` output in case of long symbol names.
- Fixed handling of paths with spaces in `gamsconfig.yaml` for items `scriptName` and `executableName` in section `solverConfig`.
- `$Call`, `$execute`, and its variants allow now spaces in the path of the script/program to be called if properly quoted.
- Fixed a zero digit `display` bug, where 0 was shown as 0.0.
- Fixed wrong matrix error detections for discrete variables and right hand side of `external equations` in certain cases.
- Fixed that the GAMS execution system corrupted control or status information in the CPUs SSE registers on Linux when using `$IfE`, `$IfThenE`, and related.

### 3.6.1.3.2 Embedded Python Code Facility

- Print the Python script line number in case an exception is raised.
- Register alias symbols in `gams.db`.
- Provide domain information for symbols via `gams.db[Sym].domains` and `gams.db[Sym].domains_as_strings`.
- Fix a problem in `pyEmbMI` for double bounded but not fixed variables.
- The Python version of GMSPython has been updated to 3.8.11.
- Removed package `schema` and its dependency `contextlib2` from GMSPython.

### 3.6.1.3.3 Extended Mathematical Programming (EMP)

- Added new stochastic EMP keyword `stageDefault` to change the default stage of symbols not assigned to a stage explicitly.
- Changed the `stage` behavior so that it does not count as a terminating error if it is used with a symbol missing in the model.

### 3.6.1.3.4 Windows Installer

- Associate `.gsp` (GAMS Studio project) files with GAMS Studio.

### 3.6.1.4 Solvers

#### 3.6.1.4.1 ANTIGONE, IPOPT, SHOT, SCIP

- Updated Intel MKL to version 2021.2.0 on Windows.

#### 3.6.1.4.2 CONVERT

- Fix default behavior if reading option file fails.
3.6.1.4.3 CPLEX

- Fixed duplicate CPLEX output in GUSS cold starts with GUSS logoption 2.
- Fixed unwanted consideration of some GAMS command line options in tuning.
- Fixed possible ignorance of an interrupt signal (Ctrl+C).
- Disabling CPLEX solution pool on default (when solnpool and solnpoolmerge are not set).
- Fixed invocation of solution pool export in case of CPLEX termination code 104: solution limit reached.
- Fixed possible crash when using mipstopexpr.

3.6.1.4.4 GUROBI

- New libraries 9.5.0.
  - Added norm constraints to general constraint type, see genconstrtype.
  - New option memlimit: Memory limit.
  - New option worklimit: Work limit.
  - New option lpwarmstart: Warm start usage in simplex.
  - New option nlpheur: Controls the NLP heuristic for non-convex quadratic models.
  - New option presos1encoding: Controls SOS1 reformulation.
  - New option presos2encoding: Controls SOS2 reformulation.
  - New option tunemetric: Metric to aggregate results into a single measure.
  - New option tunetargetmipgap: A target gap to be reached.
  - New option tunetargettime: A target runtime in seconds to be reached.
  - New option liftprojectcuts: Lift-and-project cut generation.
  - Changed default option value of TuneTrials: 0 (auto).
  - Changed default option value of CrossoverBasis: -1 (auto). The option type changed from boolean to integer.

3.6.1.4.5 GUSS

- Fixed detection of update requests with symbols that are not present in the generated model.
- Fixed incorrect behavior of option SolveEmpty (not only empty but all scenarios were skipped after threshold reached).

3.6.1.4.6 KNITRO

- Added support for MCP model type.

3.6.1.4.7 Lindo/LindoGlobal

- New libraries 13.0.309.
- Added support for function logit.
3.6.1.4.8  LocalSolver

- New libraries 10.5 (20211014).

3.6.1.4.9  Mosek

- New libraries 9.3.7.

3.6.1.4.10  Octeract

- Added new global MINLP solver Octeract 3.5.0 (https://octeract.com/).
- Octeract can be used for model types QCP, RMIQP, MIQP, NLP, DNLP, RMINLP, and MINLP and is available for Linux and Windows systems.
- For GAMS demo and community licenses, model size limitations similar to other global solvers apply. For an unlimited GAMS/Octeract license, contact sales@gams.com.

3.6.1.4.11  SCIP

- New libraries 7.0 (b2afa5403b).

3.6.1.4.12  SHOT

- New libraries 1.1.0 (11fda1ec).
  - Default value for option Subsolver.Gurobi.NumericFocus changed from 2 to 1.

3.6.1.4.13  SOPLEX

- New libraries 5.0 (ad7592b9).

3.6.1.4.14  XPRESS

- New libraries 38.01.05.

3.6.1.5  Tools

3.6.1.5.1  BIB2GMS

- We will remove the tool bib2gms in a future GAMS release.
3.6.1.5.2 FINDTHISGAMS

- Support for registry keys for .gsp (GAMS Studio project) file association has been added.

3.6.1.5.3 GAMS Studio

- New version 1.8.2.
  - New feature: Added tooltips to source code for keywords and Dollar Control Options (can be deactivated in the settings dialog).
  - Rework of the Project Explorer:
    * Changed naming from "group" to "project".
    * Files now appear in folders relative to the working directory of the assigned project.
    * Added import and export of projects.
    * The projects context menu "Project options" allows to change the name and working directory of the project.
    * Added individual working directories for each project.
    * Added dialog to set the base directory on project import.
  - Highlight remotely executed log (NEOS or GAMS Engine).
  - Removed obsolete MIRO Hypercube mode.
  - Replace TAB characters by the proper amount of spaces when pasting text.
  - Set default TAB size to 8.
  - Allow to use GAMS Engine on a HTTP server when the local SSL is not present.
  - Skipped log duplicate in the NEOS log.
  - Added new search scope Selection which allows users to search inside a text selection.
  - Added zoom for lxi tree view when zooming in lst files.
  - Stability improvements and minor bug fixes, e.g.:
    * Fixed NEOS and GAMS Engine being inactive due to an SSL detection issue.
    * Fixed links to model in NEOS log.
    * Fixed crash when editing long lines ending in an error mark.
    * Fixed highlighting of wrong results in Search Result View.
    * Fixed broken syntax highlighting of execute_*.
    * Fixed completer not opening or staying open unwanted.

3.6.1.5.4 GDX2HAR/HAR2GDX

- We will remove the tools gdx2har and har2gdx in a future GAMS release. The tools can be accessed from the CoPS web site.

3.6.1.5.5 HEXDUMP

- We will remove the tool hexdump in a future GAMS release. A similar functionality is available on Unix systems via the od utility with parameters od -A x -t xiz -v. On Windows od is distributed as part of the POSIX utilities.
3.6.1.5.6  SCENRED

- We will remove the tool SCENRED in a future GAMS release. The tool SCENRED2 should be used instead.

3.6.1.6  APIs

3.6.1.6.1  C++ high-level API

- Fixed GAMSOption::setOutput parameter being ignored in GAMSJob::run().

3.6.1.6.2  gams2numpy

- The binaries are now build with numpy 1.20.3 instead of numpy 1.19.5 for Python 3.7, 3.8, and 3.9.

3.6.1.6.3  GAMS Transfer

- New APIs GAMS Transfer Matlab and GAMS Transfer for Python to exchange data between GAMS and Matlab and Python, respectively.

3.6.1.6.4  GMD

- Allow to add symbols via gmdAddSymbol and gmdAddSymbolX of type alias (GMS_DT_ALIAS).
- Added function gmdFindSymbolWithAlias that provides the actual alias symbol rather than the aliased set in case the requested symbol is an alias.
- Added function gmdGetSymbolByNumber that provides the symbol based on the GMD_NUMBER which includes the alias symbols.
- Added information key GMD_NR_SYMBOLS_WITH_ALIAS for function gmdInfo to retrieve the number of symbols including alias symbols.

3.6.1.6.5  GMO

- Adjusted routine gmoGetMatrixCol to fail if called where quadratic structure is explicitly detected/handled (i.e. useQ is set). This routine was not intended to be used in this case.
- Added routines gmoNZ64, gmoNLNZ64, and gmoLNZ64 to get non-zero counts as 64-bit integer.
- Fixed routine gmoDirtySetRowFNLinstr: it was not computing the number of stored NL instructions correctly.

3.6.1.6.6  PAL

- Added routines palLicenseSolverCheckSizes64 and palLicenseCheck64 for solvers that use 64-bit integers for non-zero counts.
3.6.1.6.7 Python high-level API

- Fixed a problem with property `domains` of a symbol in case the domain list consists of a `GamsSet` and string elements.

3.6.1.7 Model Libraries

3.6.1.7.1 GAMS Model Library

- MS Access and Excel files have been updated to the MS Office 2007 format, i.e., files with the `xls` extension were changed to `xlsx` or `xlsm` files, files with the `mdb` extension were changed to `accdb` files.
- The structured bibliographic information has been removed. As a consequence, the `Author` column in the table of models (documentation and Studio/IDE) has been removed as well as the `Reference` section for the individual model page (web). The bibliographic references remain part of the model source.

New Models:

- `rcpsp.gms`: Resource-Constrained Project Scheduling Problem (429)

3.6.1.7.2 GAMS Test Library New models:

- `asyncfix02.gms`: Test asynchronous solves with `holdFixedAsync` setting
- `genfile01.gms`: Test generic file names
- `offinc01.gms`: Test dollar control option `$offInclude`
- `embpy11.gms`: Test proper domain info in Embedded Code
- `dco01.gms`: Test default state and switching state for dollar control options
- `epstozero1.gms`: Test writing `eps` as zero to GDX
- `fnlogit.gms`: Test correctness of `logit` intrinsic
- `empsp01.gms`: Test EMPSP keyword `stageDefault`
- `gurobi06.gms`: GUROBI test suite - general constraints norm
### 3.6.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 20.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 12.4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 10.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCTERACT 3.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 38.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.7 36 Distribution

#### 3.7.1 36.1.0 Major release (August 02, 2021)

#### 3.7.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Ricardo Manuel Pinto de Lima, Bruce McCarl, Scott
3.7.1.2 GAMS System

- The End User License Agreement (eula.pdf in the GAMS system directory) has been updated to version 08/2021.

3.7.1.2.1 GAMS

- Improved performance of model generation for certain equations where the index order of the equations caused an out of order index reference of variables like in this example:

\[ ejk(j,k).. \sum(i, x(i,j,k)) =e= 42; \]

Added new command line parameter sys19 to deactivate this performance optimization.

- Added new function platformCode and new compile time constants to be used with it.

- Added information about available CPU sockets, cores, and threads to the log.

- Improve error message for invalid values of the .scale attribute.

- Allow to read environment variables and command line parameters at execution time in put context.

- Added Put_Utility commands setEnv and dropEnv to allow environment variable modification at execution time.

- Added Python function gams.get_env to EmbeddedCode Python to access modified environment variables after Python module os has been initialized.

- Fixed a problem causing an unexpected error when beginning execution time embedded code as first statement after starting from a restart file.

- Fixed a potential warning using execution time embedded code after starting from a restart file which was created with action=c.

3.7.1.2.2 Windows Installer

- Windows installer is now built with Inno Setup 6.

- Command line parameter /allUsers=yes\|no has been deprecated and will be removed in a future GAMS release. Use the new parameter /installAllUsers=yes\|no instead.

- New command line flags /allUsers and /currentUser can be used to explicitly control if the installation is performed in administrative or non-administrative installation mode.

3.7.1.2.3 GMSPython

- The Python version has been updated to 3.8.10.

3.7.1.3 Solvers

3.7.1.3.1 Bonmin

- We plan to drop Bonmin and BonminH with one of the next major releases.
### 3.7.1.3.2 CONVERT

- Added support for quadratic constraints in formats CplexLP, CplexMPS, and FixedMPS.
- Added option `SkipNRows`.
- Added option `QExtractAlg`.
- Fixed writing `eps` for zero marginals of non-basic variables or equations in format Gams.

### 3.7.1.3.3 CPLEX

- Added the following options:
  - `multobjoptfiles`: List of option files used for individual solves within multi-objective optimization.
  - `usercut`: User cut activation.
  - `usercutpool`: Indicator to use user cuts.
  - `qextractalg`: quadratic extraction algorithm in GAMS interface.
- Fixed handling of special values (incl. missing zero values) in `multimipstart`.
- Fixed update of Jacobian elements within GUSS scenarios.
- Fixed setting `gubcovers` via `cuts`.

### 3.7.1.3.4 GUROBI

- Added option `qextractalg`: quadratic extraction algorithm in GAMS interface.

### 3.7.1.3.5 GUSS

- Added support of option `savepoint` for scenario solves.

### 3.7.1.3.6 Ipopt and IpoptH

  - Attention
    - Value `pardiso` of option `linear_solver` to choose Pardiso from Intel MKL has been renamed to `pardisomkl`. Renamed `pardiso_*` options to `pardisomkl_*`.
    - Added value `pardiso` for option `linear_solver` to choose to use Pardiso from the Pardiso Solver Project. This solver is not included in the GAMS distribution. A user needs to provide a Pardiso library. Added option `pardisolib` to specify name and path of Pardiso library. Added `pardiso_*` options.
    - Added option `hslib` to specify name and path of HSL library to be loaded at runtime (for users of GAMS/Ipopt or users that want to use HSL_MA77).
    - Changed default of `honor_original_bounds` to `no`. In addition, Ipopt now ensures that, in addition to option `bound_relax_factor`, variable bounds are relaxed by at most `constr_viol_tol`. The Ipopt log now shows the violation of variable bounds in the solution.
- Fixed that marginals for fixed variables were not computed if `fixed_variable_treatment` is set to `make_parameter` (the default). Added value `make_parameter_nodual` for this option to disable calculation of marginals for fixed variables (which can cost an extra evaluation of the Jacobian).
- Added options `max_wall_time`, `print_options_mode`, `print_advanced_options`, and `timing_statistics`.
- Ipopt should now be threadsafe also when used with MUMPS as linear solver (a mutex ensures that only one thread executes MUMPS at a time).
- For more information, including bugfixes, see the Ipopt 3.14.0 release notes.

- Changed default of `constr_viol_tol` to 1e-6.
- GAMS ResLim now sets the default for Ipopt option `max_wall_time` instead of `max_cpu_time`.
- If GAMS ScaleOpt is set, i.e., the model has been scaled manually, then the default of option `nlp_scaling_method` is set to `none`.
- The interface has been extended to warmstart Ipopt if a modified model instance is solved, e.g., in the context of GUSS. For the warmstart, previously initialized Ipopt data structures are reused if possible and option `warm_start_init_point` is enabled, see also Warmstarting IPOPT.
- Enabled `solveLink=6` capability for Ipopt and IpoptH.
- Update of HSL routines (IpoptH only):
  - MA57 3.11.0.
  - HSL_MC68 3.3.1.
  - HSL_MA86 1.6.0.
  - HSL_MA97 2.6.0.
- Update of MUMPS to version 5.4.0.
- Changed default of `mumps_pivot_order` from 5 to 7.

### 3.7.1.3.7 JAMS

- Updated behavior and logging for models with `equ.var` matches in the model statement. If the trivial (aka empty or default) model transformation is specified in the EMP info file, nothing changes: the matches are used as before. In contrast, if a nontrivial or nonempty transformation is specified, the matches are ignored when creating the reformulated model and a message to this effect is sent to the log.

### 3.7.1.3.8 KNITRO

- New libraries 12.4.
  - Improved performance on mixed-integer nonlinear problems (MINLP).
  - Improved performance when using the limited-memory quasi-Newton Hessian approximation.
  - Added option `lin_solver_max_iter`: Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro.
  - Added option `bfgs_scaling`: Specify the initial scaling to use for the BFGS or L-BFGS Hessian approximation.
  - Added option `bar_lin_sys_storage`: Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps.
  - Added option `mip_cutting_plane`: Specifies when to apply the cutting plane procedure.
  - Added option `mip_heuristic_diving`: Specifies whether or not to enable the MIP diving heuristic.
- Added option `mip_heuristic_feaspump`: Specifies whether or not to enable the MIP feasibility pump heuristic.
- Added option `mip_heuristic_mpec`: Specifies whether or not to enable the MIP MPEC heuristic.
- Added option `mip_heuristic_strategy`: Specifies the level of effort applied for the MIP heuristic search used to try to find an initial integer feasible point.
- Deprecated option `mip_heuristic`: Use `mip_heuristic_strategy` instead.

- Added option `qextractalg`: quadratic extraction algorithm in GAMS interface.

3.7.1.3.9 Lindo/LindoGlobal

- New libraries 13.0.291.

3.7.1.3.10 LocalSolver

- New libraries 10.5 (20210624).

3.7.1.3.11 MOSEK

- Added option `QEXTRACTALG`.
- New libraries 9.2.47.

3.7.1.3.12 ODHCPLEX

- New libraries 6.0.4.

3.7.1.3.13 PATH

- New libraries 5.0.04 (maintenance release) include capability to further limit preprocessor output by specifying `output.preprocess.level -1`.

3.7.1.3.14 SCIP

- New libraries 7.0 (d012dac064).
- Updated Ipopt to Ipopt 3.14.1, see also above.
3.7.1.3.15 SHOT

- New libraries 1.1 (c898dd84).
  - SHOT can now call itself to solve fixed NLP problems. Activated with new value 2 for option `Primal.FixedInteger.Solver`. For nonconvex problems it is still recommended to use an external NLP solver.
  - Added also options `Subsolver.SHOt.UseHyperplanes.Fraction`, `Subsolver.SHOt.UseHyperplanes.Use`, `Subsolver.SHOt.UseFBBT`.
  - Added support for semi-continuous and semi-integer variables.
  - Added support for special ordered sets.
  - Improved support for generating supporting hyperplanes for the entire nonlinear feasible set instead of the feasible sets for the individual constraint functions. Activated with new option `Dual.ESH.Rootsearch.UseMaxFunction`.
  - Improved support for passing nonconvex quadratic functions directly to the MIP solver (if supported).
  - Added option `ModelingSystem.GAMS.QExtractAlg`.
  - Added option `Output.GAMS.AlternateSolutionsFile`.
  - Added option `Dual.HyperplaneCuts.SaveHyperplanePoints`.
  - Added option `Dual.MIP.InfeasibilityRepair.Use`.
  - Added option `Dual.ReductionCut.Use`.
  - Changed default of option `Subsolver.Cbc.Scaling` from 0 to 4.
  - Changed default of option `Subsolver.Cplex.NumericalEmphasis` from 0 to 1.
  - Changed default of option `Subsolver.Gurobi.PoolSolution` from 1 to 10.
  - Changed default of option `Model.BoundTightening.FeasibilityBased.TimeLimit` from 5 to 2.
  - Changed default of option `Termination.DualStagnation.IterationLimit` from 50 to \( \infty \).
  - Changed meaning of values 1 and 2 for option `Model.Reformulation.Bilinear.IntegerFormulation`.

- Updated Ipopt to Ipopt 3.14.1, see also above.

3.7.1.3.16 XPRESS

- New libraries 38.01.03.
  - Xpress Optimizer Release Notes:
    * Improved MIP performance.
    * It is now possible to activate a special heuristic solve mode for MIPs, which emphasizes finding new solutions.
    * Improved sifting algorithm when used with Barrier.
    * Improved numerical stability of the Barrier algorithm.
  - Xpress Nonlinear Release Notes:
Significant MINLP performance improvements due to improved nonlinear presolver.

Improved performance on instances with very large separable nonlinear expressions due to improved automatic differentiation.

Multistart now uses the nonlinear primal integral for tie breaking.

- Added options:
  - barPerturb: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties.
  - barRefIter: After terminating the barrier algorithm, further refinement steps can be performed.
  - clamping: Allows for the adjustment of returned solution values such that they are always within bounds.
  - clamping_dual: Adjust primal slack values to always be within constraint bounds.
  - clamping_primal: Adjust primal solution to always be within primal bounds.
  - clamping_rdj: Adjust reduced costs to always be within dual bounds implied by the primal solution.
  - clamping_slacks: Adjust dual solution to always be within the dual bounds implied by the slacks.
  - genConsAbsTransformation: Specifies the reformulation method for absolute value general constraints at the beginning of the search.
  - heurEmphasis: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap.
  - inputTol: Tolerance on input values elements.
  - mipComponents: Determines whether disconnected components in a MIP should be solved as separate MIPs.
  - mipConcurrentNodes: Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled.
  - mipConcurrentSolves: Selects the number of concurrent solves to start for a MIP.
  - mipRestartFactor: Fine tune initial conditions to trigger an in-tree restart.
  - mipRestartGapThreshold: Initial gap threshold to delay in-tree restart.
  - netStallLimit: Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex.
  - nodeProbingEffort: Adjusts the overall level of node probing.
  - outputControls: Toggles the printing of all control settings at the beginning of the search.
  - ppFactor: Partial pricing candidate list sizing parameter.
  - preFolding: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition.
  - sifoPasses: Determines how quickly we allow to grow the worker problems during the sifting algorithm.
  - sifoSwitch: Determines which algorithm to use for solving the subproblems during sifting.
  - xslp_presolveOps_noLinear: Avoid performing linear reductions at the NLP level.
  - xslp_presolveOps_noSimplifier: Avoid simplifying nonlinear expressions.

- Added option values:
  - deterministic: 2.
  - mipRestart: 2.

- Changed option defaults:
  - mipPresolve: -257 to -1.
  - mipPresolve_symmetryReductions: 0 to 1.

- Deprecated options:
  - xslp_matrixTol. Use matrixTol instead.

- Removed options:
• Added option `algAfterNetwork`: Algorithm to be used for the clean up step after the network simplex solver.

• Added option `qextractalg`: quadratic extraction algorithm in GAMS interface.

• Removed options that were marked deprecated in GAMS 35.

3.7.1.4 Tools

3.7.1.4.1 Cholesky, Eigenvalue, Eigenvector, Invert

• We will remove the tools `cholesky`, `eigenvalue`, `eigenvector`, `invert` in a future GAMS release. Similar functionality has been added via `$libInclude linalg {cholesky,eigenvalue,eigenvector,invert}`. See the header of the file in `<sysdir>\inclib\linalg.gms` for details.

3.7.1.4.2 GAMS Studio

• New version 1.7.2.
  
  – Dropped support for Ubuntu 16.04 LTS and openSUSE Leap 15.2 because the updated AppImage now requires glibc 2.27. Dropped support for Debian 9.
  
  
  – Improved visual distinction between current and other groups in Project Explorer.
  
  – Added a warning message before printing large files.
  
  – Added a list of solvers for each model type to be selected in Extended Parameter Editor and Configuration Editor.
  
  – Added shortcut `Alt+I` (Mac OS X: `Option+I`) to fold blocks of dollar control options.
  
  – Added setting to automatically collapse dollar control option blocks when opening a file (default: off).
  
  – Changed shortcuts to show/hide Extended Parameter Editor
    
    * Windows/Linux: From `Ctrl-Alt-3` to `Shift+Ctrl+3`.
    * Mac OS X: From `Shift-Command-3` to `Control-Option-3`.
  
  – Improvements for syntax highlighter and code completer, e.g.:
    
    * Added support for system attributes and compile-time constants.
    * Added support of quoted elements in syntax highlighter.
    * Disabled code completer for certain code blocks like `$onEmbeddedCode` or `$onPut`.
  
  – Improvements for remote execution support, e.g.:
    
    * Added support for GAMS Engine server with self-signed certificate (`https`).
    * Added support for GAMS Engine server with `http` protocol.
    * Added option to remember login credentials for GAMS Engine in Settings.
    * Renamed "MIRO" Settings page to "Remote" to better reflect new content.
* Improved GAMS Engine dialog.
* Added file type EFI to send additional files to a GAMS Engine server.

- Stability improvements and minor bug fixes, e.g.:
  * Fixed duplicate file entries in the Project Explorer when having two references with different case to a single file
  * Fixed Studio title bar appearing outside of screen in some cases.
  * Fixed wrong content of "File" tab in Reference File Viewer.
  * Fixed missing check of some data locations for GAMS license file.
  * Fixed ESC key not behaving like close button in Settings dialog.
  * Fixed crash when reopening a closed log tab.
  * Fixed that MIRO menu entries were not enabled when MIRO was automatically discovered.
  * Fixed that theme could be broken in Settings dialog when using cancel button.

### 3.7.1.4.3 GDXRank, GDXRename

- We will remove the tools gdxrank and gdxrename in a future GAMS release. Similar functionality has been added via $libInclude gdxservice \{gdxrank,gdxservice\}. See the header of the file in <sysdir>\inclib\gdxservice.gms for details.
- The $libInclude rank functionality now uses $libInclude gdxservice gdxrank to sort a one dimensional parameter instead of the gdxrank tool.

### 3.7.1.4.4 MCFilter

- We will remove the tool mcfilter in a future GAMS release. In case you depend on this tool, this discussion at GAMS World Forum gives an alternative.

### 3.7.1.4.5 MSAppAvail, Shellexecute, XLISTalk

- We will remove the tools msappavail, shelleexecute, xlistalk in a future GAMS release. Similar functionality has been added via $libInclude win32 \{msappavail,shelleexecute,xlistalk\}. See the header of the file in <sysdir>\inclib\win32.gms for details.

### 3.7.1.5 APIs

- As announced we removed all Delphi variants but the dcp version (i.e. gdxdcpdef.pas) and all Fortran variants but the f9 version (i.e. gdxf9def.f90) of the low-level APIs.

#### 3.7.1.5.1 CFG

- The function cfgAlgAllowsModifyProblem now reports correctly for all solvers if they provide the xyzModifyProblem functionality instead of a only for a fixed (and incorrect) list of solvers.
3.7.1.5.2 C++ high-level API

- Removed dependency on Qt. Now requiring C++17.
- As announced we dropped support for Microsoft Visual Studio 2015.
- As announced, the Clang compiled version of libgamscpp.dylib does not work on Mac OS X 10.14 or older anymore. It has been moved to the subfolder apifiles/C++/lib/Clang/. The new libgamscpp.dylib in apifiles/C++/lib/ is compiled with GCC.

3.7.1.5.3 gams_engine

- Updated for GAMS Engine version 21.06.09 and now using OpenAPI Generator version 5.1.1.

3.7.1.5.4 gams2numpy

- New optional parameter merge in gams2numpy.gmdFillSymbolStr that allows duplicate records in the numpy array as well as writing to non-empty symbols.
- New optional parameter relaxedType in gams2numpy.gmdFillSymbol(Str|Raw), and gams2numpy.gdxWriteSymbol(Str|Raw) that allows to automatically convert the columns of the numpy array into the required data types if possible.
- Fixed a bug that prevented error messages from being shown on Linux and further improved error handling.

3.7.1.5.5 GAMSX

- As announced, we have dropped the GAMSX API with this release. The example xp_example2 for various languages has been adjusted to call the GAMS executable instead of the gamsxRunExecDLL GAMS API function.

3.7.1.5.6 GEV

- The gmoptr argument in function gevGetCurrentSolver is deprecated and not used anymore. It will be removed from this function in some future release. Moreover, the function should only be used in solver link library functions xyzReadyAPI or xyzCallSolver.
- Add GEV option SavePoint to query the savePoint option.

3.7.1.5.7 GMO

- Added option qExtractAlg to control the technique used to extract quadratic structure from GAMS models. This structure extraction happens automatically for any QCP model type (e.g. QCP, MIQCP) and the current default choice ThreePass (= 1) is almost always the best one. But there some cases where DoubleForward (= 2) performs much better.
3.7.1.5.8 Solverlink library interface

- The API of a solver link library has changed. This will only affect users that install additional solvers or build and maintain own solver link libraries. The older version of the API is deprecated and will be removed in a future GAMS release. Moreover, COIN-OR project GAMSlinks provides a template to work with both new and old APIs.

3.7.1.6 Model Libraries

3.7.1.6.1 GAMS API Library

- Updated models CPPex2, CSex2, Cex2, Fex2, Jex1, Jex2, PBuildXPLLevelAPI, and VBex2 to adjust for the removal of the GAMSX API.
- Updated model Pgams_engine to be compatible with the latest gams_engine API.

3.7.1.6.2 GAMS Data Library

- Many models have been adjusted to use the library include win32 to compensate for the future drop of the utilities msappavail and shellexecute.
- Updated models CHP, CHP2, Portfolio, Samurai, SpawnGAMSEExcel, Sudoku, and transxls. The corresponding Excel files have been changed from format .xls to .xlsm. Moreover, the VBA code has been adjusted to work without the dropped GAMSX API. The registry key to find the GAMS system directory has been updated.
- Model Samurai2 has been dropped since it became a duplicate of model Samurai after the removal of the GAMSX API.
- Model LeastSquares has been updated to use $libInclude linalg OLS for solving the linear regression problem. The model also works now with more descriptive data set (diabetes data).

3.7.1.6.3 GAMS Test Library  New models:

- platform01.gms : Test platform function
- qcp12.gms : Test different GMO Q extraction algorithms
- asyncfix01.gms : Test asynchronous solves with holdFixed
- savep2.gms : Test savepoint with async solves
- convert16.gms : CONVERT test suite - Basic test of convert output for quadratic model
- embpy09.gms : Test Embedded Code after restart
- scenso18.gms : Check if solver with modify problem use hotstart in GUSS
- gdxrename1.gms : Simple gdxrename test
- gdxrename2.gms : Tests some gdxrename stuff
- scenso9.gms : Test option savepoint in GUSS
- embpy10.gms : Test Embedded Code for Environment Variables
- etsuf01.gms : Test Execution Time Suffixes
- emp33.gms : JAMS: matches from model statement may be ignored

Models trilib01, trilib02, trilib03, parlib01, cpplib00:

- Removed jinja2 from associated source archive. It is now required to have the Python package jinja2 installed to run the ql.py script.
### 3.7.1.7 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 20.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 12.4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 10.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 38.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.7.2 36.2.0 Minor release (September 03, 2021)

#### 3.7.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Daniel Dias, Andreas Lundell, Bruce McCarl, and Tom Rutherford.

#### 3.7.2.2 GAMS System

##### 3.7.2.2.1 GAMS
• Added new command line parameter `holdFixedAsync` to allow the use of `holdFixed` for models solved asynchronously. Activating this option can lead to inconsistent solutions being reported.

• Fixed a potential crash introduced with GAMS 36.1.0 related to the improved performance of model generation.

3.7.2.3 Solvers

3.7.2.3.1 CONVERT

• Fixed MCP mapping for DumpGDX.

3.7.2.3.2 GAMSCHK

• Fix default solve behavior: if no GCK file is specified, no subsolver should be called as no solution is wanted.

3.7.2.3.3 GUSS

• Fixed options `OptfileInit` and `Optfile` being ignored.
• Added fallback to cold start in case of hot start failure.

3.7.2.3.4 IPOPT

• New libraries 3.14.3.
• Updated MUMPS to version 5.4.1.

3.7.2.3.5 MOSEK

• New libraries 9.3.4.

3.7.2.3.6 SHOT

• New libraries 1.1 (9a8893e7).

3.7.2.4 Tools

3.7.2.4.1 GAMS Studio

• New version 1.7.3 with various bug fixes, stability improvements, and minor enhancements, e.g.:
  – Allow to duplicate multiple lines at once.
  – Added navigation and selection by word using `ctrl+left/right` in LOG-Viewer.
  – Fixed highlighting issue when a symbol declaration ends with a space.
  – Fixed horizontal scroll in LST viewer when cursor is out of view.
  – Fixed that searching backwards in modified document was not working.
  – Fixed "Replace" ignoring case sensitivity option.
3.8 35 Distribution

3.8.1 35.1.0 Major release (April 29, 2021)

3.8.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Matthew Adams, Jürgen Apfelbeck, Wolfgang Britz, Yacine Gaoua, Carolin Kellenbrink, Masoud Javadi, Bruce McCarl, Scott McDonald, Will Raymond, Tom Rutherford, Sebastian Spieker, and Kirk A. Yost.

3.8.1.2 Platforms

- macOS 11 (on Intel CPUs) has been added to the list of supported Mac OS X versions.

3.8.1.3 GAMS System

3.8.1.3.1 GAMS

- Added new dollar control options $gdxLoad and $gdxUnload to load and unload symbols from and to GDX files in a single instruction at compile time.
- Print an extra log line for echo dollar control options $echo, $echoN, and $onEcho, which write to an external file. When running GAMS from one of the GAMS IDEs, this log line can be clicked to open the file created.
- Extended the report generated by dmpSym and dmpUserSym to include another column reporting a memory estimate for each set, parameter, variable, and equation.
- Removed the columns DIM-OK, ACCESS, SPECVAL, EXTERN, TABLE, DOMAIN, and LAGLEAD from the report generated by dmpUserSym. This information is very technical and mostly of internal interest. These columns are still available with dmpSym.
- The status of Multi-Threading for the Grid and Multi-Threading Solve Facility has been changed from beta to released and fully supported.
- Fixed that the stars option was ignored for some messages in the output file.
- Fixed a problem that causes default records in the data statement or when loaded from GDX (see $load) being ignored with implicit set definition. So with

```
Set i;
Parameter p(i<) / i0 0, i1 1, i2 2 /;
```

only i1 and i2 made it into i. Now, i0 is added as well.
- Fixed a problem, where $onDotL and $onDotScale were ignored in put statements.
- Fixed a problem, where $onDelim was applied incorrectly to display and option statements.
- Fixed a problem, where $onSymXRef and $onUElXRef were ignored when there were too many references.
- Fixed and extended the behavior of $eval: If a GAMS function was hidden by a user defined symbol, $eval ignored the latter. This has been changed. Now, the GAMS function needs to be prefixed with system. to be accessed in this case. An example for this behavior is given here.
- Fixed a potential problem when doing a filtered load of symbols with domain violations through EmbeddedCode.
- Fixed an unwanted compilation error that happened if $onExternalInput or $onExternalOutput were used on the same line and after $if.
- Fixed a problem where dumpOpt in combination with $onVerbatim skipped a leading substitution character in an $ifThen block.
3.8.1.4 Solvers

3.8.1.4.1 AlphaECP

- New version 2.11.01.
  - New options:
    - ECPtoltype: AlphaECP termination criterion.
    - MIPoptcltype: Strategy to increase MIPoptclim.
    - MIoptimaliter: MIP is solved to optimality with this frequency.
    - TOLoptcr: Relative termination tolerance for MINLP.
  - Minor bugfixes.
  - For more information, see the release paper.

3.8.1.4.2 ANTIGONE, IPOPT, SHOT, SCIP

- Updated Intel MKL to version 2021.2.0 on Linux and Mac OS X.

3.8.1.4.3 BARON

- The NLP solver selected with option extNLPSolver is used in case NLPSol is set to -1 or 6 (and AllowExternal is 1). In the previous releases Conopt was used with NLPSol value -1.

3.8.1.4.4 BONMIN

- Fixed an issue where a problem with SOS constraints was rejected due to falsely detected negative variable bounds.

3.8.1.4.5 CONOPT

- New libraries for CONOPT3: 3.17K
  - Fixed perturbations of the Jacobian for external equations where 2nd order derivatives are not available.
- Fixed an issue resulting in insufficient CONOPT3 memory.
- Fixed an issue with multi-threading in CONOPT 4.

3.8.1.4.6 CPLEX

- New libraries 20.1.0.1.
- Added option usercallparmfile: Command-line parameter include file used in GAMS command-line calls triggered by BCH.
- Fixed incorrect reporting of infeasible solutions of the unrelaxed model in feasopt mode.
- Fixed incorrect variable matching for options objnreltol and objnabstol.
- Fixed parsing of individual objective functions for multiobjective optimization (any ordering of equation definitions is allowed now).
3.8.1.4.7 GAMSCHK

- Fix problem with handling of row/column patterns introduced in GAMS 34.3.

3.8.1.4.8 GUROBI

- New libraries 9.1.2.
- Added possibility to write a single GDX file with the solution pool (see solnpooldump).
- Added the following options:
  - solnpooldump: Enable merged solution pool GDX.
  - solnpooldumpnumsym: Number of variable symbols in merged solution pool GDX.
  - solnpooldumpprefix: Prefix for variable symbols in merged solution pool GDX.
  - miptrace: Filename of MIP trace file (formerly hidden and undocumented).
  - miptraceon: Node interval when a trace record is written (formerly hidden and undocumented).
  - miptraceoff: Time interval when a trace record is written (formerly hidden and undocumented).
- Fixed incorrectly processed (hidden) options isvname and appname.

3.8.1.4.9 GUSS/Scenario Solver

- Measure time internally rather then relying on solvers' reported resUsd to determine the overall running time.

3.8.1.4.10 KNITRO

- New GAMS/Knitro link using the KN interface.
- Enabled the possibility to use solverlink = 5 on Linux (other platforms were already enabled).
- Added support of variable and equation scaling.
- Added options:
  - names: Enable to pass variable and equation names to Knitro.
  - blasoptionlib: Specifies a dynamic library name that contains object code for BLAS/LAPACK functions.
  - bndrange: Specifies max limits on the magnitude of constraint and variable bounds.
  - cg_pmem: Specifies the amount of nonzero elements per column of the Hessian of the Lagrangian which are retained when computing the incomplete Cholesky preconditioner.
  - cg_precond: Specifies whether an incomplete Cholesky preconditioner is applied during CG iterations in barrier algorithms.
  - cg_stoptol: Specifies the relative stopping tolerance used for the conjugate gradient (CG) subproblem solves.
  - convex: Declare the problem as convex by setting to 1 or non-convex by setting to 0.
  - cpuplatform: Specify the target instruction set architecture.
  - initpenalty: Specifies the initial penalty parameter used in the Knitro merit functions.
- **linesearch**: Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate.

- **newpoint**: Specifies additional action to take after every iteration in a solve of a continuous problem.

- **presolve_level**: Set the level of presolve operations to enable through the Knitro presolver.

- **presolve_initpt**: Control whether the Knitro presolver can shift a user-supplied initial point.

- **presolve_passes**: Set a maximum limit on the number of passes through the Knitro presolve operations.

- **presolveop_tighten**: Determine whether or not to enable the Knitro presolve operation to tighten variable bounds.

- **restarts**: Specifies whether or not to enable automatic restarts in Knitro.

- **restarts_maxit**: When restarts are enabled, this option can be used to specify a maximum number of iterations before enforcing a restart.

- **strat_warm_start**: Specifies whether or not to invoke a warm-start strategy.

- **findiff_reltol**: Specifies the relative stepsize used for finite-difference gradients.

- **findiff_terminate**: Specifies the termination criteria when using finite-difference gradients.

- **infeasitol_iters**: The optimization process will terminate if the relative change in the feasibility error is less than infeasitol for infeasitol_iters consecutive infeasible iterations.

- **xtol_iters**: The optimization process will terminate if the relative change in all components of the solution point estimate is less than xtol for xtol_iters.

- **bar_conic_enable**: Enable special treatments for conic constraints when using the Interior/Direct algorithm (has no affect when using the Interior/CG algorithm).

- **bar_initpi_MPEC**: Specifies the initial value for the MPEC penalty parameter $\pi$ used when solving problems with complementarity constraints using the barrier algorithms.

- **bar_linsys**: Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps.

- **bar_maxcorrectors**: Specifies the maximum number of corrector steps allowed for primal-dual steps.

- **bar_slackboundpush**: Specifies the amount by which the barrier slack variables are initially pushed inside the bounds.

- **bar_switchobj**: Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase.

- **act_Lpfeastol**: Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms.

- **act_Lppenalty**: Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms.

- **act_Lppresolve**: Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms.

- **act_parametric**: Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms.

- **act_qppenalty**: Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm.

- **mip_clique**: Specifies rules for adding clique cuts.

- **mip_cutfactor**: This value specifies a limit on the number of cuts added to a node subproblem.

- **mip_heuristic_terminate**: Specifies the condition for terminating the MIP heuristic.

- **mip_intvar_strategy**: Specifies how to handle integer variables.

- **mip_mir**: Specifies rules for adding mixed integer rounding cuts.

- **mip_selectdir**: Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch and bound tree.
3.8 35 Distribution

- **mip_zerohalf**: Specifies rules for adding zero-half cuts.
- **ma_outsub**: Enable writing algorithm output to files for the multi-algorithm (algorithm=5) procedure.
- **ms_num_to_save**: Specifies the number of distinct feasible points to save in a file named `knitro_mspoints`.
- **ms_outsub**: Enable writing algorithm output to files for the parallel multistart procedure.
- **ms_savetol**: Specifies the tolerance for deciding if two feasible points are distinct.
- **par_mnumthreads**: Specify the number of threads to use for multistart (when `ms_enable = 1`).
- **out_csvinfo**: Controls whether or not to generates a file `knitro.solve`.
- **out_csvname**: Use to specify a custom csv filename when using `out_csvinfo`.
- **out_hints**: Specifies whether to print diagnostic hints (e.g. about user option settings) after solving.
- **outappend**: Specifies whether output should be started in a new file, or appended to existing files.
- **outdir**: Specifies a single directory as the location to write all output files.
- **outmode**: Specifies where to direct the output from Knitro.
- **outname**: Use to specify a custom filename when output is written to a file using `outmode`.

- Added values to the following options:
  - **blasoption**: 2 (dynamic)
  - **bar_pencons**: -1 (auto), 3 (equalities)
  - **hessopt**: 7 (gauss_newton)
  - **honorbnds**: -1 (auto)
  - **ma_terminate**: 3
  - **mip_knapsack**: 3 (all)
  - **mip_outlevel**: 2 (iterstime), 3 (root)
  - **mip_rootalg**: 4 (sqp), 5 (multi)
  - **scale**: 2 (user_none), 3 (internal)

- Updated the following options to KNITRO defaults:
  - **fstopval**: maxdouble
  - **honorbnds**: -1 (auto)
  - **par_blasnumthreads**: 0
  - **par_ismnumthreads**: 0
  - **xtol**: 1e-12
  - **bar_pencons**: -1 (auto)
  - **bar_switchrule**: -1 (auto)
  - **mip_maxnodes**: 0
  - **mip_maxsolves**: 0
  - **mip_rounding**: -1 (auto)

- Deprecated options:
  - **bar_maxbacktrack**: Use `linesearch_maxtrials` instead.
  - **maxcgit**: Use `cg_maxit` instead.
  - **pivot**: Use `linsolver_pivottol` instead.

- Removed formerly deprecated options:
  - **maxcrossit**
• Removed unused option `reform`.
• Fixed an issue with options not being forwarded to Knitro.

3.8.1.4.11 Lindo/LindoGlobal

• New libraries 13.0.262:
  • LP Solver Improvements:
    * With new enhancements made to the simplex solvers, the average performance on large instances where the number of variables is several times larger than the number of constraints has increased by several folds compared to the previous version.
    * Improved performance on LP’s when using multiple cores with concurrent execution of Primal, Dual, and Barrier.
  • MIP (Mixed Integer Program) Solver:
    * Improved selection of defaults for cuts and heuristics.
  • Global Solver Improvements:
    * Highly improved ability to recognize convexity of various composite functions, especially involving logarithms and sums, which can give a performance improvement of one order-of-magnitude in these cases.
    * Performance improvement for convex-concave type functions.
    * Enhanced capabilities for converting nonlinear functions to linear forms.
    * Substantial performance improvement for non-convex quadratic models.

3.8.1.4.12 LocalSolver

• New libraries 10.0 (20210330).
• We plan to drop LocalSolver and LocalSolver70 by the end of 2021.

3.8.1.4.13 MOSEK

• New libraries 9.2.41.

3.8.1.4.14 MPSGE

• Increased the internal MAXFUN limit (from 5000 to 10000) on the number of components (inputs, outputs, taxes, etc.) in any MPSGE row.

3.8.1.4.15 ODHCPLEX

• New libraries 5.34.
3.8.1.4.16 SBB

- Added option `usercallparmfile`: Command-line parameter include file used in GAMS command-line calls triggered by BCH.

3.8.1.4.17 SCIP

- New libraries 7.0 (c53330372e).

3.8.1.4.18 SNOPT

- New libraries 7.7.7 and new GAMS/SNOPT link.
- Changed option value to SNOPT default for the following options:
  - Linesearch Tolerance: 0.9
  - LI Singularity Tolerance: 3.2e-11
- Removed the following parameters:
  - Start Objective Check
  - Start Constraint Check
  - Stop Objective Check
  - Stop Constraint Check
  - Derivative Level

3.8.1.4.19 XPRESS

- Added support for choosing a different algorithm than primal simplex for solving the fixed MIP (see `mipCleanup`). The algorithm can be chosen with `defaultAlg` and `lpFlags`.
- Deprecated the following options:
  - `algorithm`: Use `defaultAlg` or `lpFlags` instead.
  - `extraPresolve`
  - `lpThreads`
- Added the following options:
  - `algAfterCrossover`: Algorithm to be used for the final clean up step after the crossover.
  - `autoPerturb`: Simplex: Indicates whether automatic perturbation is performed.
  - `autoScaling`: Whether the Optimizer should automatically select between different scaling algorithms.
  - `backtrackTie`: Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed.
  - `barCores`: If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm.
  - `barFailIterLimit`: Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm.
  - `barFreeScale`: Defines how the barrier algorithm scales free variables.
  - `barGapTarget`: Newton barrier: Target tolerance for the relative duality gap.
- **barKernel**: Newton barrier: Defines how centrality is weighted in the barrier algorithm.
- **barObjScale**: Defines how the barrier scales the objective.
- **barOrderThreads**: If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton barrier method.
- **barPresolveOps**: Controls the Newton barrier specific presolve operations.
- **barRegularize**: Determines how the barrier algorithm applies regularization on the KKT system.
- **barRhsScale**: Defines how the barrier scales the right hand side.
- **branchChoice**: Once a global entity has been selected for branching, this control determines which of the branches is solved first.
- **branchDisj**: Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search.
- **branchStructural**: Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search.
- **cacheSize**: Newton barrier: L2 or L3 (see notes) cache size in kB (kilobytes) of the CPU.
- **choleskyAlg**: Newton barrier: Type of Cholesky factorization used.
- **choleskyTol**: Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm.
- **conflictCuts**: Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts.
- **coresPerCpu**: Used to override the detected value of the number of cores on a CPU.
- **cpuTime**: How time should be measured when timings are reported in the log and when checking against time limits.
- **crossoverAccuracyTol**: Newton barrier: Determines how crossover adjusts the default relative pivot tolerance.
- **crossoverIterLimit**: Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates.
- **crossoverOps**: Newton barrier: Bit vector for adjusting the behavior of the crossover procedure.
- **cutFactor**: Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during global search.
- **cutSelect**: Bit vector providing detailed control of the cuts created for the root node of a global solve.
- **dualGradient**: Simplex: Dual simplex pricing method.
- **dualize**: Whether presolve should form the dual of the problem.
- **dualizeOps**: Bit-vector control for adjusting the behavior when a problem is dualized.
- **dualPerturb**: Factor by which the problem will be perturbed prior to optimization by dual simplex.
- **dualStrategy**: Bit-vector control specifies the dual simplex strategy.
- **elimFillin**: Amount of fill-in allowed when performing an elimination in presolve.
- **elimTol**: Markowitz tolerance for the elimination phase of the presolve.
- **feasibilityPump**: Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node.
- **feasTolPerturb**: Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes.
- **feasTolTarget**: Target feasibility tolerance for the solution refiner.
- **forceParallelDual**: Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm.
- **genConsDualReductions**: Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs.
- **heurBeforeLp**: Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved.

- **heurDiveIterLimit**: Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic.

- **heurDiveRandomize**: Level of randomization to apply in the diving heuristic.

- **heurDiveSoftRounding**: Enables a more cautious strategy for the diving heuristic.

- **heurDiveSpeedUp**: Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed.

- **heurDiveStrategy**: Branch and Bound: Chooses the strategy for the diving heuristic.

- **heurForceSpecialObj**: Branch and Bound: Whether local search heuristics without objective or with an auxiliary objective should always be used, despite the automatic selection of the Optimizer.

- **heurFreq**: Branch and Bound: Frequency at which heuristics are used in the tree search.

- **heurSearchEffort**: Adjusts the overall level of the local search heuristics.

- **heurSearchFreq**: Branch and Bound: How often the local search heuristic should be run in the tree.

- **heurSearchRootCutFreq**: How frequently to run the local search heuristic during root cutting.

- **heurSearchRootSelect**: Bit vector control for selecting which local search heuristics to apply on the root node of a global solve.

- **heurSearchTreeSelect**: Bit vector control for selecting which local search heuristics to apply during the tree search of a global solve.

- **heurStrategy**: Branch and Bound: Heuristic strategy.

- **historyCosts**: Branch and Bound: How to update the pseudo cost for a global entity when a strong branch or a regular branch is applied.

- **indLinBigM**: Indicator constraints can be internally converted to regular rows (i.e. linearized) using a BigM coefficient whenever the BigM coefficient is smaller or equal to this value.

- **indPreLinBigM**: During presolve, indicator constraints can be internally replaced with regular rows (i.e. linearized) using a BigM coefficient whenever the BigM coefficient is smaller or equal to this value.

- **l1Cache**: Newton barrier: L1 cache size in kB (kilo bytes) of the CPU.

- **lnpBest**: Number of infeasible global entities to create lift-and-project cuts for during each round of Gomory cuts at the top node (see gomCuts).

- **lnpIterLimit**: Number of iterations to perform in improving each lift-and-project cut.

- **localChoice**: Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree.

- **lpFlags**: Bit-vector control which defines the algorithm for solving an LP problem or the initial LP relaxation of a MIP problem.

- **lpFolding**: Simplex and barrier: Whether to fold an LP problem before solving it.

- **lpLogDelay**: Time interval between two LP log lines.

- **lpLogStyle**: Simplex: Style of the simplex log.

- **lpRefineIterLimit**: Simplex iteration limit the solution refiner can spend.

- **markowitzTol**: Markowitz tolerance used for the factorization of the basis matrix.

- **maxCutTime**: Maximum amount of time allowed for generation of cutting planes and reoptimization.

- **maxImpliedBound**: Presolve: When tighter bounds are calculated during MIP preprocessing, only bounds whose absolute value are smaller than maxImpliedBound will be applied to the problem.

- **maxLocalBacktrack**: Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node.
- **maxMemoryHard**: Sets the maximum amount of memory in megabytes the optimizer should allocate.
- **maxMemorySoft**: When resourceStrategy is enabled, this control sets the maximum amount of memory in megabytes the optimizer targets to allocate.
- **maxMipTasks**: Branch-and-Bound: The maximum number of tasks to run in parallel during a MIP solve.
- **maxScaleFactor**: Determines the maximum scaling factor that can be applied during scaling.
- **maxTime**: Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time.
- **mipDualReductions**: Branch and Bound: Limits operations that can reduce the MIP solution space.
- **mipFracReduce**: Branch and Bound: Specifies how often the optimizer should run a heuristic to reduce the number of fractional integer variables in the node LP solutions.
- **mipKappaFreq**: Branch and Bound: Specifies how frequently the basis condition number (also known as kappa) should be calculated during the branch-and-bound search.
- **mipRampUp**: Controls the strategy used by the parallel MIP solver during the ramp-up phase of a branch-and-bound tree search.
- **mipRefineIterLimit**: Defines an effort limit expressed as simplex iterations for the MIP solution refiner.
- **mipRestart**: Branch and Bound: Controls strategy for in-tree restarts.
- **mipToltarget**: Target mipTol value used by the automatic MIP solution refiner as defined by refineOps.
- **miqcpAlg**: Determines which algorithm is to be used to solve mixed integer quadratically constrained and mixed integer second order cone problems.
- **xslp_mipAlgorithm**: Bitmap describing the MISLP algorithms to be used.
- **xslp_mipFixStepBounds**: Bitmap describing the step-bound fixing strategy during MISLP.
- **xslp_mipRelaxStepBounds**: Bitmap describing the step-bound relaxation strategy during MISLP.
- **netCuts**: Determines the addition of multi-commodity network cuts to a problem.
- **objScaleFactor**: Custom global objective scaling factor, expressed as a power of 2.
- **optimalityTolTarget**: Target optimality tolerance for the solution refiner.
- **outputLog**: Controls the level of output produced by the Optimizer during optimization.
- **outputTol**: Zero tolerance on print values.
- **preAnalyticCenter**: Determines whether analytic centers should be computed and used for variable fixing and the generation of alternative reduced costs.
- **preBasisRed**: Determines whether a lattice basis reduction algorithm should be attempted as part of presolve.
- **preBndRedCone**: Determines whether second order cone constraints should be used for inferring bound reductions on variables when solving a MIP.
- **preBndRedQuad**: Determines whether convex quadratic constraints should be used for inferring bound reductions on variables when solving a MIP.
- **preCoeffElim**: Presolve: Specifies whether the optimizer should attempt to recombine constraints in order to reduce the number of non zero coefficients when presolving a mixed integer problem.
- **preComponents**: Determines whether small independent components should be detected and solved as individual subproblems during root node processing.
- **preComponentsEffort**: Presolve: Adjusts the overall effort for the independent component presolver.
- **preConeDecomp**: Presolve: Decompose regular and rotated cones with more than two elements and apply Outer Approximation on the resulting components.
- **preConvertSeparable**: Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints.

- **preDomCol**: Presolve: Determines the level of dominated column removal reductions to perform when presolving a mixed integer problem.

- **preDomRow**: Presolve: Determines the level of dominated row removal reductions to perform when presolving a problem.

- **preDupRow**: Presolve: Determines the type of duplicate rows to look for and eliminate when presolving a problem.

- **preElimQuad**: Presolve: Allows for elimination of quadratic variables via doubleton rows.

- **preImplications**: Presolve: Determines whether to use implication structures to remove redundant rows.

- **preLinDep**: Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem.

- **preObjCutDetect**: Presolve: Determines whether to check for constraints that are parallel or near parallel to a linear objective function, and which can safely be removed.

- **presolveMaxGrow**: Limit on how much the number of non-zero coefficients is allowed to grow during presolve, specified as a ratio of the number of non-zero coefficients in the original problem.

- **presolveOps**: Specifies the operations which are performed during the presolve.

- **presolvePasses**: Number of reduction rounds to be performed in presolve.

- **primalPerturb**: Factor by which the problem will be perturbed prior to optimization by primal simplex.

- **primalUnshift**: Determines whether primal is allowed to call dual to unshift.

- **qcCuts**: Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for the root node, when solving a mixed integer quadratically constrained or mixed integer second order conic problem with outer approximation.

- **qcRootAlg**: Determines which algorithm is to be used to solve the root of a mixed integer quadratically constrained or mixed integer second order cone problem, when outer approximation is used.

- **qSimplexOps**: Controls the behavior of the quadratic simplex solvers.

- **quadraticUnshift**: Determines whether an extra solution purification step is called after a solution found by the quadratic simplex (either primal or dual).

- **randomSeed**: Sets the initial seed to use for the pseudo-random number generator in the Optimizer.

- **refineOps**: Specifies when the solution refiner should be executed to reduce solution infeasibilities.

- **relaxTreeMemoryLimit**: When the memory used by the branch and bound search tree exceeds the target specified by the `treeMemoryLimit` control, the optimizer will try to reduce this by writing nodes to the global file.

- **repairIndefInitEq**: Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible.

- **resourceStrategy**: Controls whether the optimizer is allowed to make nondeterministic decisions if memory is running low in an effort to preserve memory and finish the solve.

- **rootPresolve**: Determines if presolving should be performed on the problem after the global search has finished with root cutting and heuristics.

- **sbBest**: Number of infeasible global entities to initialize pseudo costs on each node.

- **sbEffort**: Adjusts the overall amount of effort when using strong branching to select an infeasible global entity to branch on.

- **sbEstimate**: Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible global entity to branch on.
- **sbIterLimit**: Number of dual iterations to perform in strong branching for each entity.
- **sifting**: Determines whether to enable sifting algorithm with the dual simplex method.
- **xslp_algorithm**: Bitmap describing the SLP algorithm(s) to be used.
- **xslp_analyze**: Bitmap activating additional options supporting model / solution path analysis.
- **xslp_augmentation**: Bitmap describing the SLP augmentation method(s) to be used.
- **xslp_barStartOps**: Controls behaviour when the barrier is used to solve the linearizations.
- **xslp_convergenceOps**: Bitmap describing which convergence tests should be carried out.
- **xslp_filter**: Bitmap for controlling solution updates.
- **xslp_presolveOps**: Bitmap indicating the SLP presolve actions to be taken.
- **xslp_zeroCriterion**: Bitmap determining the behavior of the placeholder deletion procedure.
- **sosRefTol**: Minimum relative gap between the ordering values of elements in a special ordered set.
- **treeCompression**: When writing nodes to the global file, the optimizer can try to use data-compression techniques to reduce the size of the global file on disk.
- **treeCutSelect**: Bit vector providing detailed control of the cuts created during the tree search of a global solve.
- **treeMemoryLimit**: Soft limit, in megabytes, for the amount of memory to use in storing the branch and bound search tree.
- **treeMemorySavingTarget**: When the memory used by the branch-and-bound search tree exceeds the limit specified by the treeMemoryLimit control, the optimizer will try to save memory by writing lower-rated sections of the tree to the global file.
- **treeQCCuts**: Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for nodes other than the root node, when solving a mixed integer quadratically constrained or mixed integer second order conic problem with outer approximation.

- Added options to conveniently set bits of XPRESS bitmap options:
  - New bits of **refineOps**:
    - refineOps_lpOptimal
    - refineOps_mipSolution
    - refineOps_mipNodeLp
    - refineOps_lpPresolve
    - refineOps_iterativeRefiner
    - refineOps_refinerPrecision
    - refineOps_refinerUsePrimal
    - refineOps_refinerUseDual
    - refineOps_mipFixGlobals
    - refineOps_mipFixGlobalsTarget
  - New bits of **mipPresolve**:
    - mipPresolve_reducedCostFixing
    - mipPresolve_logicPreprocessing
    - mipPresolve_allowChangeBounds
    - mipPresolve_dualReductions
    - mipPresolve_globalCoeffTightening
    - mipPresolve_objBasedReductions
    - mipPresolve_allowTreeRestart
    - mipPresolve_symmetryReductions
  - New bits of **presolveOps**:
    - presolveOps_singletonColRemoval
    - presolveOps_singletonRowRemoval
* presolveOps_forcingRowRemoval
* presolveOps_dualReductions
* presolveOps_redundantRowRemoval
* presolveOps_duplicateColRemoval
* presolveOps_duplicateRowRemoval
* presolveOps_strongDualReductions
* presolveOps_variableEliminations
* presolveOps_noIpReductions
* presolveOps_noGlobalDomainChange
* presolveOps_noAdvIpReductions
* presolveOps_linDependRowRemoval
* presolveOps_noIntVarEliminations
* presolveOps_noIntVarAndSosDetect

- New bits of scaling:
  * scaling_rowScaling
  * scaling_colScaling
  * scaling_rowScalingAgain
  * scaling_maximum
  * scaling_curtisReid
  * scaling_byMaxElemNotGeoMean
  * scaling_bigM
  * scaling_simplexObjScaling
  * scaling_ignoreQuadRowPart
  * scaling_beforePresolve
  * scaling_noScalingRowsUp
  * scaling_noScalingColsDown
  * scaling_disableGlobalObjScaling
  * scaling_rhsScaling
  * scaling_noAggressiveQScaling
  * scaling_slackScaling

- New bits of lpFlags:
  * lpFlags_dual
  * lpFlags_primal
  * lpFlags_barrier
  * lpFlags_network

- New bits of cutSelect:
  * cutSelect_clique
  * cutSelect_mir
  * cutSelect_cover
  * cutSelect_mirRowAggregation
  * cutSelect_flowpath
  * cutSelect_implication
  * cutSelect_liftAndProject
  * cutSelect_disableCutRows
  * cutSelect отделCover
  * cutSelect_zeroHalf
  * cutSelect_indicator

- New bits of treeCutSelect:
  * treeCutSelect_clique
  * treeCutSelect_mir
• Renamed the following options to match the original XPRESS options names. Old option names currently serve as alias, but are deprecated and will be removed in an upcoming release:
  - `treePresolveKeepBasis` to `treePresolve_keepBasis`.
  - `slpKnitroOptFile` to `knitroOptFile`.
  - All XPRESS Nonlinear (SLP) options, e.g., `slpSolver` to `xslp_solver`.

• Fixed default values of single bit options for XPRESS bitmap options with negative default values.

3.8.1.5 Tools

3.8.1.5.1 GAMS Studio

• New version 1.6.1.
  - New feature: Static Code Completion which can be activated by pressing `Ctrl + Space`.
  - New feature: Allow to move multiple lines up and down using `Ctrl + Shift + Arrow`.
  - New feature: Allow to close view tabs using the middle mouse button.
  - Added instant update of tab size in editor when changing corresponding setting.
  - Added word under cursor as default search entry when no text selection is present.
  - Allow to move block selections using the arrow keys.
  - Added `*.inc` as common extension for GAMS files.
  - Added user defined extensions for GAMS files.
  - Added automatic reloading of files that appear in the GAMS log.
  - Added setting for user defined extensions for files to be automatically reloaded.
  - Added auto-reload button to file-changed dialog.
  - Added Studio Documentation entry in Help menu.
  - Added setting to change default open file behavior (new group or current group) and adjusted menu entry for alternative file opening behavior.
  - Hotkey `F1` jumps to the corresponding documentation entry for Project Explorer, Process Logs, GDX Diff Dialog, and Search Widget.
  - Improved GDX Viewer, e.g.:
    * Added facility to access domains and filters in Table View.
    * Show original index position of domains in both list view and table view.
    * Improved automatic column widths.
  - Stability improvements and minor bug fixes, e.g.:
    * Fixed crash when LST file is missing (e.g. because of setting `output=NUL`).
    * Fixed "Reset View" not resetting splitter in the LST/LXI-Viewer.
    * Fixed crash related to comment shortcut in solver option editor when pressing it multiple times.
* Fixed wrong sort order after resetting GDX Viewer.
* Fixed GAMS engine not appending /api to the URL on Mac OS X when clicking OK directly after editing the URL.
* Fixed `eolCom` not working in some dollar control option lines (like `$include`).
* Fixed jump to next search result if the result is in a different file and the current one has no results.
* Fixed jump to search result occasionally not working.
* Fixed Find Next/Prev behavior for `.opt` files.

### 3.8.1.5.2 GDXDUMP

- Added a new line at the end of the file, which was missing if the option `noHeader` is set.

### 3.8.1.5.3 GDXMERGE

- Added command line option `strict`. When set to `true`, `gdxmerge` will terminate with an error in case input files cannot be found or the output file already exists.

### 3.8.1.5.4 GDXXRW

- Fixed a potential missing initialization if `incRC` is set for symbols with either `cDim` or `rDim` equal 0.

### 3.8.1.6 APIs

#### 3.8.1.6.1 CFG

- With the next major release the last argument `opt: pointer` of function `cfgAlgReadyAPI` will be dropped.

#### 3.8.1.6.2 DCT

- Functions `dctLoadWithHandle`, `dctSymDomIdx`, `dctDomNameCount`, and `dctDomName` have been deprecated and will be removed in the near future.

#### 3.8.1.6.3 gams2numpy

- Improved error handling.
- New optional parameter `domains` for `gdxWriteSymbolStr` and `gdxWriteSymbolRaw` that can be used to specify the domains of the symbol. Parameter needs to be a list of strings.
3.8.1.6.4 GMO

- Updated to better handle cases where quadratic structure is explicitly detected (i.e. useQ is set) and some but not all of the nonlinear equations are quadratic. Several new functions have been added. In addition, some existing functions have been renamed - the new names are more descriptive and/or more consistent with new and existing function names. The old names are still available as deprecated synonyms - we expect to remove these synonyms in the near future and potentially without warning.

  - Updated API version to 20 - compatible with some older API versions.
  - New read-only properties:
    * gmoObjLNZ: Number of linear nonzeros in objective gradient.
    * gmoObjNLNZEx: Number of GMOORDER_NL nonzeros in objective gradient.
    * gmoObjjQNZeX: Number of GMOORDER_Q nonzeros in objective gradient.
    * gmoObjCVecNZ: Number of nonzeros in c vector of objective (-1 if Q information not used).
    * gmoObjConstEx: Objective constant - this is independent of useQ.
    * gmoObjQConst: Constant in solvers quadratic objective.

  - New functions:
    * gmoGetObjCVec: Get c vector of quadratic objective.
    * gmoGetObjSparseEx: Get information for gradient of objective function (sparse).
    * gmoGetRowStatEx: Get Jacobian row nonzero counts: total and by GMOORDER_XX.
    * gmoGetRowCVecNZOne: Number of nonzeros in c vector of row si (-1 if Q information not used).
    * gmoGetRhsOneEx: Get individual equation RHS - independent of useQ.
    * gmoGetRowSparseEx: Get info for one row of Jacobian (sparse).
    * gmoGetRowCVec: Get c vector of the quadratic form for one row.
    * gmoGetRowQConst: Get the constant of the quadratic form for one row.

  - New read-write properties:
    * gmoHessInclQRows: if useQ is true, this boolean property toggles the inclusion of GMOORDER_Q rows in the Hessian.

  Attention

    The default was originally true but has accidentally been changed to false with this release (35.1.0). That is, if useQ is set, then the Hessian does not include entries from quadratic equations anymore. The default is changed back to true in GAMS 38.2.0.

  - Renamed properties/functions:
    * gmoObjQNZ → gmoObjQMatNZ: Number of nonzeros in lower triangle of Q matrix of objective (-1 if useQ is false).
    * gmoGetObjQ → gmoGetObjQMat: Get lower triangle of Q matrix of objective.
    * gmoGetRowQ → gmoGetRowQMat: Get lower triangle of Q matrix of one row.

3.8.1.6.5 High-Level APIs

- We might drop support for macOS 10.14 for the C++ API with the next major release.
- Fixed an issue with an updater symbol that tries to update a non-existing symbol in the model instance. This used to make the solve method of GAMSModelInstance fail, but now just increases the no-match count.
- Fixed GAMSOptions.Output (.NET), GAMSOptions.setOutput (Java), and GamsOptions.output (Python) parameter being ignored in GAMSJob::Run().
3.8.1.6.6 Low-Level APIs

- For the next major releases, some clean-up of the distributed APIs is planned. In particular, it is planned to remove all Delphi variants but the dcp version (i.e. gdxcpdef.pas) and all Fortran variants but the f9 version (i.e. gdxf9def.f90). Please contact us, if that causes any trouble for your operation.

3.8.1.6.7 PAL

- Added functions palDataDirs and palConfigDirs to retrieve standard locations searched by GAMS.

3.8.1.7 Model Libraries

3.8.1.7.1 GAMS Model Library

- nurses.gms: A Nurse Scheduling Problem (428)

3.8.1.7.2 GAMS Test Library New models:

- gdxmerg3.gms: Test strict mode
- unload15.gms: Test $gdxUnload
- load13.gms: Test $gdxLoad
- conopt02.gms: CONOPT test suite - multi-thread test
- implset2.gms: Test for Implicit Set Definition with default records
- prevwork1.gms: PreviousWork Test
- decla3.gms: Test correct loading after multi declaration of domain set
- delim5.gms: Limited scope for $onDelim
- load14.gms: Test loading of GDX file with bad UELs
- eval08.gms: Test $eval with hidden functions
- embpy07.gms: Test no_match_limit for model instance
- embpy08.gms: Test filtered load from Embedded Code
### 3.8.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 20.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.13</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 12.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 10.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 36.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.8.2 35.2.0 Minor release (June 02, 2021)

#### 3.8.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Johannes Hedtrich, Bruce McCarl, Evangelos Panos, Thomas Rutherford, and Kirk A. Yost.
3.8.2.2 GAMS System

3.8.2.2.1 GAMS

- Allow to write point files with asynchronous solves as well.
- Moved the log line pointing to a reference file from the end of the compilation phase to the end of the run, to make it easier to recognize.
- Automatically deactivate holdFixed if a model is solved asynchronously, since this could lead to inconsistent solutions otherwise. Note that this can make a square model non-square as in cns01. Also note, that this could require some adjustment to models which actually relied on the fact, that models solved like this did not update the solution for fixed variables. Such a change was done for the model jacobi.
- Fixed a problem where the file path specified with $gdxIn or $gdxLoad was ignored when inputDir was applied.
- Fixed a potentially broken work file written with previousWork.
- Fixed the default for command line parameter empty. It should be on but was off.
- Fixed the dollar control options $setComps, $setNames, and $splitOption: They should generate scoped compile-time variable but generated local ones.
- Fixed potentially wrongly reported bounds for models solved with solveLink=6.
- Fixed missing info about objective variable with solveLink = 3 and asyncSolLst = 1.

3.8.2.2.2 EMP

- Fixed handling of the empty UEL in the EMP info file

3.8.2.2.3 GMSPython

- Removed package tqdm.

3.8.2.3 Solvers

3.8.2.3.1 AlphaECP

- New library with some minor bug fixes.

3.8.2.3.2 CONVERT

- Fixed exporting of variable scale attribute in DumpGDX format.

3.8.2.3.3 CPLEX

- Fixed incorrect model status for feasible models after time out in multi-objective optimization.
3.8.2.3.4 KNITRO

- Fixed incorrect model status for KNITRO return code "All nodes have been explored. Integer feasible point found." (KN_RC_MIP_EXH_FEAS).

3.8.2.4 Tools

3.8.2.4.1 GAMS Studio

- New version 1.6.2. with various bug fixes, stability improvements and minor enhancements, e.g.:
  - Fixed issues with enumerated parameters in extended parameter editor.
  - Fixed issues with "set", "option", and "table" in code-completer.
  - Fixed code-completer opening in non-code files.
  - Fixed "Open in current group" not being used for "User Model Library" and "New file".

3.8.2.5 APIs

3.8.2.5.1 Python

- Fixed a bug in the GDX expert-level API that required gdxSymbolGetDomain to be called in a read context (e.g. after gdxDataReadStrStart).
- Fixed a bug in gams2numpy functions gdxReadSymbolStr and gdxReadSymbolRaw that altered the case of the symName argument.

3.8.2.5.2 DCT

- Fixed broken function dctSymDomNames for symbols that have the universe as part of their domain. This caused a crash in sensitivity analysis of CPLEX and Gurobi for models with such variable or equation symbols.

3.8.2.6 Model Libraries

3.8.2.6.1 GAMS Model Library

- Changed jacobi.gms : Asynchronous Jacobi Methods to work with the changed holdFixed behavior with asynchronous solves, mentioned above.

3.9 34 Distribution

3.9.1 34.1.0 Major release (January 29, 2021)

3.9.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Wolfgang Britz, Michael Ferris, Erwin Kalvelagen, Antti Lehtila, Bruce McCarl, Scott McDonald, Cornelius Rüther, Shima Sasanpour, Kaushik Sinha, Elizabeth Wong, and Qi Yuan.
3.9.1.2 Platforms

- The minimal required GLIBC version of the distribution for GNU/Linux is now 2.17.
- Mac OS X 10.13 (High Sierra) is no longer supported.

3.9.1.3 GAMS System

3.9.1.3.1 GAMS

- Improved performance of *dense* operations with out-of-order index positions, as in the following code snippet, where the assignment to \( d_1 \) was significantly slower than the one to \( d_2 \) in the past:

  \[
  d_1(j,i) = \sum ij(i,j), \exp(pij(i,j)))
  \]

  \[
  d_2(j,i) = \sum ij(i,j), \abs(pij(i,j)))
  \]

- Added new debugging option `dmpUserSym`, to print all symbols defined by the user.
- Added new variants `$showFiles`, `$showMacros`, and `$showVariables` of `$show` for shorter reports.
- Print an extra log line for compile time GDX operations `$gdxIn` and `$gdxOut`. When running GAMS from one of the GAMS IDEs, these log line can be clicked to open the file referenced.
- Add anchors to the log that can be clicked when running GAMS from one of the GAMS IDEs for each mention of an input/include file.
- Added extra info to the log in case of an unmatched `$ifThen` and its variants, that can be clicked when a GAMS IDE is used to jump to the initial dollar control option.
- Added more details about the problem to the log in case of an error analyzing the solve statement.
- Added new variants for the command line parameter `fileCase` which allow to use upper or lower case for referenced filenames only and not for the path to the files.
- Fixed a bug where certain checks for errors in the model structure were skipped, if upper and lower bound of a variable are the same. While the selected solver would already have rejected the problem most likely, now GAMS generates an execution error also if lower and upper bounds are the same and
  - the bounds of discrete variables do not have integer values,
  - the lower and upper bounds of binary variables are not equal to 0 or 1, or,
  - the lower bounds of semiCont/semiInt variables are smaller than 0.
- Altered the behavior of `execute.checkErrorLevel`, `putUtility exec.checkErrorLevel`, and `putUtility shell.checkErrorLevel` so the execution would stop right away also within a Flow Control structure, like a `loop`, if an error is encountered. In the past, it would just stop after executing the flow control structure completely.
- Changed the behavior for unloading symbols with `$unload`. In previous versions the writing of the symbols happened while the compiler processed the `$unload` instructions. With this version forward the writing of the symbols and data to GDX is delayed until the GDX file is about to be closed. This has several advantages, e.g. writing of aliases and full domain information, but it can result in different content of the GDX file in some corner cases. Please check for details in the documentation.
- Avoid export of predefined symbols diag and sameAs to GDX. When all symbols should be exported, these are skipped. When they are tried to be exported explicitly, there will be a compilation error.
- Allow to use `Put Utility save` to write a save file of the current state of execution inside loops and other flow control structures as well.
- Fixed a wrongly raised compilation error which could happen when an implicit set definition was used with a `table statement` without data.
- Fixed a problem where `dumpOpt` in combination with `$onVerbatim` skipped parts of an `$ifThen` block.
3.9.1.3.2 GMSPython

- Added the packages `pyexcelerate`, `pyyaml`, `xlsxwriter`, and their dependencies.

3.9.1.4 Solvers

3.9.1.4.1 AMPL

- As announced, dropped the meta-solver AMPL.

3.9.1.4.2 BARON

  - Improved interface to IPOPT for local search.
  - Changed default for option LPSol to new value -1, which makes BARON use CPLEX for LP or MIP relaxations, if licensed, and CLP/CBC otherwise.
  - Further performance improvements and bugfixes.
- Added option BarName to specify name of file into which to write problem in BARON format.

3.9.1.4.3 BDMLP

- As announced, dropped the solver BDMLP.

3.9.1.4.4 BENCH

- As announced, dropped the tool BENCH.

3.9.1.4.5 CONOPT

- New library 4.22.
  - Some improvements for models with external equations.
3.9.1.4.6 CONVERT

- The two solver links CONVERT and CONVERTD were merged under the name CONVERT. Features that were formerly available in only one of the two solver links are now all usable with the new CONVERT link (except for the dropped features, see below). CONVERTD is an alias for CONVERT now.

- JuMP format:
  - Added constraint names.
  - Using `@NLconstraint` and `@NLobjective` instead of `add_NL_constraint` and `set_NL_objective`, respectively.
  - Changed default of option JuMP to `jump.jl`.

- Pyomo format:
  - Fix export of SOS1 and SOS2 variable types.
  - Changed default of option Pyomo to `pyomo.py`.

- DumpGDX format (formerly Hessian or Jacobian):
  - The current behaviour of disabling variable and equation names or UELs when setting the option to "NOVENAMES.gdx" or "NOUELS.gdx" has no special effect anymore. Use GDXNames and GDXUELs instead.

- New options:
  - Width
  - DumpGDX (replaces options Hessian and Jacobian)
  - GDXNames (instead of setting the former options Hessian or Jacobian to "NOVENAMES.gdx")
  - GDXUELs (instead of setting the former options Hessian or Jacobian to "NOUELS.gdx")
  - GDXHessian (enables hessian information for DumpGDX)
  - GDXQuadratic (enables quadratic information for DumpGDX)

- Options PermuteEqus and PermuteVars are now available for all target languages.

- Removed the following formats:
  - AlphaECP
  - AmplNLC
  - Analyze
  - AnalyzeS
  - Baron (GAMS/Baron link provides the option BarName to export this format)
  - Lgo
  - LindoMPI (GAMS/Lindo link provides the option WRITEMPI to export this format)
  - LocalSolver (GAMS/Localsolver link provides the option writelsp to export this format)
  - LSPSol
  - Memo
  - Minopt
  - SFS
  - ViennaDag

- Removed the following experimental formats:
  - AmplNLCG
  - CHull
- CppAD
- Lago
- NLP2MCPE
- PDCO
- Qmaker (use DumpGDX with GDXQuadratic enabled instead)
- UnitBC

- The following formats are deprecated and may be removed in an upcoming release:
  - Hessian (use DumpGDX with GDXHessian enabled instead)
  - Jacobian (use DumpGDX instead)

- The following options don't have any effect anymore and will be removed in an upcoming release:
  - ConeReform
  - IntervalEvalDebug
  - Match
  - Terminate

3.9.1.4.7 CPLEX

- New libraries 20.1.0.
  - Conflict Analysis (iis) now supports SOS variables and indicator constraints.
  - Added options:
    - * prereform*: allows to set presolve reformulations
    - * conflictalg*: algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model
    - * nodedcuts*: allows to decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree
    - * sos1reform*: automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)
    - * sos2reform*: automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)
    - * Value 5 to mipemphasis*: finding high quality feasible solutions as early as possible
  - Deprecated options:
    - * prelinear*: Use prereform instead
  - Deprecated features (will be removed in a future CPLEX release):
    - Remote Object functionality.
    - Distributed parallel optimization of mixed integer programming models (MIP).

- Fixed error messages for CPLEX error 5002 (CPXERR_Q_NOT_POS_DEF).
- Fixed an error that caused a crash for special option file input.
- Fixed an error that wrote additional quotes in GDX files of ranging and column generation information.
- Fixed miptrace timing info.
3.9.1.4.8 Gurobi

- New libraries 9.1.1.
- Added the following options:
  - SolFiles: location to store intermediate solution files
  - DualReductions: disables dual reductions in presolve
  - ProjImpliedCuts: projected implied bound cut generation
  - PSDCuts: PSD cut generation
  - FeasRelaxBigM: Big-M value for feasibility relaxations
- Added possible option values for the following options:
  - MinRelNodes: -1 (automatic)
  - NodeMethod: -1 (automatic)
  - PumpPasses: -1 (automatic)
  - StartNodeLimit: -3 (shut off)
  - ZeroObjNodes: -1 (automatic)
  - AggFill: -1 (automatic)
  - PreSparsify: -1 (automatic)
  - TuneResults: -1 (best results)
  - IISMethod: 2 (ignores bound constraints)
  - IISMethod: 3 (IIS the LP relaxation)
  - Method: 5 (both primal and dual simplex)
- Set option value to GUROBI default for the following options:
  - scaleflag: -1
  - Cutoff: maxdouble
  - ImproveStartGap: 0
  - MinRelNodes: -1
  - NodeMethod: -1
  - PartitionPlace: 15
  - PumpPasses: -1
  - ZeroObjNodes: -1
  - AggFill: -1
  - PreSOS2BigM: -1
  - PreSparsify: -1
  - TuneResults: -1
  - TuneTrials: 3
- Removed the following options:
  - premiqpmethod
  - workerport
- Options for how to access the Gurobi compute server, cluster manager, or Instant Cloud can all be set through the Gurobi license file. Therefore, the following link options have been removed:
  - computeserver
  - csrgroup
- cspassword
- csport
- cspriority
- csrouter
- cstimeoutfrac
- cstlsinsecure
- icsecretkey
- icpool
- icpriority
- instantcloud

- Fixed issue of missing logfile output for logoption = 4.
- Fixed an error that wrote additional quotes in GDX files of ranging information.

3.9.1.4.9 Ipopt

- Updated MUMPS to version 5.3.5.

3.9.1.4.10 KESTREL

- The environment variable NEOS_EMAIL can now be used to provide an e-mail address when submitting a job to NEOS.
- Fixed unintentional reading from option file.

3.9.1.4.11 KNITRO

- New libraries 12.3.0.
  - Improvements in performance, memory usage, numerical stability, and overall stability.

3.9.1.4.12 LINGO

- As announced, dropped the meta-solver LINGO.
3.9.1.4.13 **LS** As announced, dropped the tool LS.

One possibility to adapt existing GAMS code is to use the Python function `numpy.linalg.lstsq` via the GAMS embedded code facility.

For example, the GAMS code

```gams
Set i;
Parameter x(i), y(i);
Variables b0, b1, sse;
Equations sumsq, fit(i);
sumsq.. sse =n= 0;
fit(i).. y(i) =e= b0 + b1*x(i);
option lp = ls;
model leastsq /fit,sumsq/;
solve leastsq using lp minimizing sse;
```

to compute a linear regression for \((x,y)\) can be replaced by

```gams
Set i;
Parameter x(i), y(i);
Scalars b0, b1, sse;
embeddedCode Python:
  import numpy as np
  x = np.array(gams.get('x', keyFormat=KeyFormat.SKIP))
  y = np.array(gams.get('y', keyFormat=KeyFormat.SKIP))
  A = np.vstack([np.ones(len(x)), x,]).T
  res = np.linalg.lstsq(A, y, rcond=None)
  gams.set('b0', [res[0][0]])
  gams.set('b1', [res[0][1]])
  gams.set('sse', [res[1][0]])
endEmbeddedCode
```

For a more detailed example, see the new `LeastSquares.gms` model from the GAMS Data Library.

3.9.1.4.14 **Mosek**

- New libraries 9.2.35.

3.9.1.4.15 **MPECDUMP**

- As announced, dropped the tool MPECDUMP.

3.9.1.4.16 **NLPEC**

- Set the (hidden) GAMS option `relPath` to 1 for the execution of the scalar GAMS model. This fixes a problem when running NLPEC under KESTREL on NEOS.
3.9.1.4.17  ODHCPLEX

- New libraries 5.31.
- Fixed missing output of final CPLEX solve for fixed model.
- Fixed possible values of option `odhpresolve` to 1 and 2.

3.9.1.4.18  PYOMO

- As announced, dropped the meta-solver PYOMO.

3.9.1.4.19  SCIP

- New libraries 7.0 (9a11615b47).

3.9.1.4.20  SoPlex

- New libraries 5.0.2.

3.9.1.4.21  XPRESS

- Non-convex QCPs don't have to be solved as NLP anymore, but are solved by XPRESS Nonlinear if a XPRESS-NLP or XPRESS-MINLP license is used (XPRESS-MINLP for MIQCPs).
- Fixed an issue that prevented the use of XPRESS Knitro with a XPRESS link license.
- Fixed a bug that prevented to solve (MI)QCP models with an XPRESS-MIP license.

3.9.1.5  Tools

3.9.1.5.1  CSV2GDX

- Report the line/field records to the log that lead to `UNDF` conversion for trace level 1 or bigger.
3.9.1.5.2 GAMS Studio

- New version 1.5.2.
  - New feature: Allow to edit and store custom color themes in the settings dialog.
  - New feature: Allow to define custom sections in the code, that can be folded using $onFold / $offFold.
  - Improved MIRO deploy dialog.
  - Added e-mail field for NEOS submissions in NEOS dialog. This can also be initialized from environment variable `NEOS_EMAIL` (which can be set permanently in gamsconfig.yaml).
  - Changed "Autosave on Run" setting to save ALL modified files, not just files of the current group.
  - Changed default light theme comment color to grey for better distinction from declarations.
  - Disabled GAMS syntax highlighting between $onPut/$offPut and $onEcho/$offEcho.
  - Added "select entry" on clicking timestamp in log.
  - Pressing `Enter` in the LXI tree scrolls to the corresponding position in the LST view.
  - Adjusted list of files in Reference File Viewer to be sorted by order in which they have been used.
  - Stability improvements and minor bug fixes, e.g.:
    - Fixed rare problem with undetected external file changes.
    - Fixed broken include file links in combination with end of line comments.
    - Fixed missing request for new search cache after switching file.
    - Fixed rare crash related to LOG output.
    - Fixed endlessly recurring message box for invalid GDX file.
    - Fixed missing update of parameter history for a file assigned to more than one group.
    - Fixed search results not being highlighted in `.log` files.
    - Fixed search performance problem after reset search parameters.
    - Fixed behavior of ESC key when current widget has a selection.
    - Fixed syntax highlighting of `table` statements not detecting row headers.
    - Fixed print dialog shortcut not working directly after Studio launch.

3.9.1.5.3 GDX2VEDA

- Increase the maximum length of symbol names in the Veda definition file from 31 to 63.

3.9.1.5.4 GDXDUMP

- When using the option noData, empty symbols were still written with a data statement. Now empty symbols are also loaded from GDX.

3.9.1.5.5 GDXMRW

- Switch from C to C++ as the compiler for these utilities.
- An unordered_map container is now used to address a performance issue with rgdx and filtered reads.
3.9.1.6 APIs

3.9.1.6.1 C++

- We plan to drop support for Microsoft Visual Studio 2015 for the C++ API with the next major release.

3.9.1.6.2 Python

- Added support for Python 3.9.
- gams2numpy:
  - New functions `gdxGetUelList` and `gmdGetUelList` allow to retrieve the list of UELs.
  - `gdxReadSymbolRaw` and `gmdReadSymbolRaw` do not return the list of UELs anymore.
  - `gdxReadSymbolStr` and `gmdReadSymbolStr` have a new optional input parameter `uelList`.
  - Fixed a bug that prevented loading of required shared objects in certain cases.

3.9.1.6.3 Matlab

- New GAMS Matlab API
  - The API is distributed as package GAMS located in `apifiles/Matlab/api`.
  - Examples can be found in `apifiles/Matlab/examples`.

3.9.1.6.4 GMO

- Fixed the functions `gmoGetObjSparse`, `gmoGetRowSparse`, `gmoGetColSparse`, `gmoHessLoad`, `gmoHessLagStruct`, and `gmoHessLagVal`: they now work properly when `forceLinear` is set, e.g., when using GUSS.

3.9.1.7 Model Libraries

3.9.1.7.1 GAMS API Library

- Changed the following examples to use the MONO compiler `mcs` instead of `dmcs` on Unix:
  - `CSBenders.gms`: Test object oriented C# API using a (multi-threaded) Benders Decomposition Algorithm
  - `CSCalcInverse.gms`: Test expert level C# API to read and write indexed GDX
  - `CSCutstock.gms`: Test object oriented C# API using a cutting stock example
  - `CSex1.gms`: Test expert level C# API to read and write GDX
  - `CSex2.gms`: Test expert level C# API to read and write GDX, set options and execute GAMS
  - `CSWarehouse.gms`: Test object oriented C# API using a warehouse location problem
3.9.1.7.2 GAMS Data Library  New models:

- LeastSquares.gms : Demonstrate the use of numpy.linalg.lstsq on the diabetes test problem using linalg ols (141)

3.9.1.7.3 GAMS Test Library  New models:

- show01.gms : Test $show feature
- dmpsym01.gms : Test dmpSym and dmpUserSym feature
- ide01.gms : Basic test for IDE mode
- cplex08.gms : CPLEX test suite - conflict analysis (841)
- convert13.gms : CONVERT test suite - apply dict to gams format (842)
- convert14.gms : CONVERT test suite - nonlinear expression reformulations (843)
- convert15.gms : CONVERT test suite - support of nonlinear functions (844)
- unload12.gms : Make sure that sameAs and diag cannot be exported to GDX (845)
- unload13.gms : Unload with relaxed and full domain (846)
- unload14.gms : Check that unload happens at GDX file closing time (847)
- dumpopt1.gms : Test verbatim dumpOpt (848)

Removed models:

- LS01 (394)
- LS02 (395)
- LS03 (397)
- LS04 (398)

3.9.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th></th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 20.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.9.2 34.2.0 Minor release (February 05, 2021)

#### 3.9.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz and Bruce McCarl.

#### 3.9.2.2 Solvers

##### 3.9.2.2.1 CONOPT

- New libraries 4.23.
  - Improvements for models with external equations, models with very large objective terms, and models with variables that only appear in the objective function.
- Fix parallel function evaluation.

##### 3.9.2.2.2 CONVERT

- Fix variable coefficients in JuMP format.

##### 3.9.2.2.3 CPLEX

- Fixed issue of non-working option `userinebicall`.

---

<table>
<thead>
<tr>
<th>Solver</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPOPT 3.13</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLER 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 36.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
3.9.2.2.4 GAMSCHK

- The output formatting for the BLOCKPIC and MATCHIT procedures was adjusted to better handle very small or large values.

3.9.3 34.3.0 Minor release (February 25, 2021)

3.9.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Luis Alberto Herrero Rozas, Robert Howlett, Erwin Kalvelagen, Antti Lehtila, Bruce McCarl, and Shima Sasanpour.

3.9.3.2 GAMS System

3.9.3.2.1 GAMS

- Fixed a problem with embedded Python code and directory names longer than 128 characters.
- Fixed embedded Python code exceptions not handled correctly in case the exception message is empty.
- Fixed a problem where dumpOpt in combination with $onVerbatim produced wrong dollar conditions in an $ifThen block.
- Fixed an abnormal termination issue that occurred on Windows with pathologically-structured large QP models and solvers that use QP structure explicitly.
- Fixed a potentially broken work file written with previousWork. There was a problem introduced with GAMS 34.1 for models written to the work file.
- Fixed wrong behavior of break in certain loops over a set with a single element.

3.9.3.3 Solvers

3.9.3.3.1 CONOPT

- New libraries 4.24:
  - The pre-processor has been updated so it will always return a solution.
- Fixed incorrect reference of row / column in error message.

3.9.3.3.2 CONVERT

- Fixed possibly incorrect sign of objective function in formats with explicit objective function (e.g., Ampl or JuMP).
- Fixed missing objective constant in formats with explicit objective function (e.g., Ampl or JuMP).
- Fixed possibly missing variable / equation in formats GAMS, DictMap and DumpGDX when converting multiple formats at a time.
- Fixed possibly missing parenthesis when dealing with minus zero.
3.9.3.3 GAMSCHK

- An error in the POSTOPT procedure that could lead to a segmentation fault or undefined behavior was fixed.
- The output formatting in the BLOCKLIST procedure was improved.

3.9.3.4 MOSEK

- New libraries 9.2.38.

3.9.3.5 ODHCPLEX

- Fixed issue of possibly incorrect objective bound reported by GAMS.

3.9.3.6 OsiMosek

- Fixed reported dual bound (ObjEst) and model status if solved without having closed the optimality gap.

3.9.3.4 Tools

3.9.3.4.1 GAMS Studio

- New version 1.5.4. with various bug fixes, stability improvements and minor enhancements, e.g.:
  - Added support for ports in GAMS Engine URL.
  - Improved visibility of current word and parentheses highlighting.
  - Fixed wrong color initialization for dark theme.
  - Fixed unwanted cursor jump when searching listing files without results.
  - Fixed MIRO execution issue if model filenames contain spaces.
  - Fixed GDX file reloading while being rewritten.
  - Fixed file links in log not handling relative paths on Windows.

3.9.3.4.2 GDXDUMP

- With option noData the empty symbols are marked with an end of line comment `!!empty`.
- The dollar control options are written in a consistent camel-case style if the output is in GAMS syntax.

3.9.3.4.3 GDXXRW

- Fixed a crash which happened when the rng was defined multiple times for a text by accident.
3.9.3.6  Expert Level APIs

- When introducing `gdxStoreDomainSets` with GAMS 33 the default was accidentally set to 0 (don’t enable use of one dimensional sets as domain sets). This has been corrected with this version. Setting `gdxStoreDomainSets` to 0 is for real experts only, since it can cause inconsistent GDX files when used in combination with `gdxSymbolSetDomain`.

3.9.3.6.1  Matlab

- Fixed Java class attribute access to support Matlab versions down to 2017b.

3.10  33 Distribution

3.10.1  33.1.0 Major release (November 01, 2020)

3.10.1.1  Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Pablo Cachinero, Michael Ferris, and Ben Huebner.

3.10.1.2  Platforms

- We plan to increase the minimal required GLIBC version of the distribution for GNU/Linux to 2.17 with the next major release.
- We plan to drop support for Mac OS X 10.13 (High Sierra) with the next major release.

3.10.1.3  GAMS System

3.10.1.3.1  GAMS

- Allow to set the `.stage` attribute for discrete variables as well. This can be useful for certain solvers like SCIP.
- The status of the Embedded Code Facility has been changed from beta to released and fully supported. The nature of embedded code (especially Python embedded code) prohibits a level of backward compatibility as we guarantee for intrinsic GAMS language features: The Python interpreter together with distributed packages are evolving. Moreover, the GAMS/Python communication (e.g. via `gams.get()`/`set()`) will be subject to necessary modifications.
- Allow multi-line Python strings constructed with triple quotes (""" or """") in embedded Python code without getting additional spaces from the internal extra indentation. For multiple lines of Python code that use the line continuation character \\, extra indentation is not omitted.
- Added new dollar control options `$on/offIDCProtect` to control if symbols which are declared as external input can be changed at execution time.
• Quoted text in a display statement adds now an entry to the navigation pane of the LST Viewer in GAMS Studio as well.

• Print an extra log line when execute_load (or one of its variants) is executed. When running GAMS from one of the GAMS IDEs, this log line can be clicked to open the file loaded. Also the log lines generated for execute_unload[idx] were adjusted for better distinction. So adding these lines to a model

```plaintext
execute_unload  't.gdx', f;
execute_unloadidx 't.gdx', f;
execute_load    't.gdx', f;
```

generates this log:
--- GDX File (execute_unload) C:\t m p\t.gdx
--- GDX File (execute_unloadidx) C:\t m p\t.gdx
--- GDX File (execute_load) C:\t m p\t.gdx

• Fixed a problem using captureModelInstance with solveLink=5/6/7; if captureModelInstance=1 and solveLink=6/7, solveLink will be reset to 3/4 automatically.

• Fixed a problem with file names longer than 128 characters for option IDCGenerateJSON.

3.10.1.3.2 GMSPython

• Added the packages urllib3, certifi, and their dependencies.

3.10.1.4 Solvers

3.10.1.4.1 AMPL, LINGO, MPECDUMP, PYOMO

• We plan to drop the meta-solvers AMPL, LINGO, MPECDUMP, and PYOMO with the next major release. Convert will continue to produce model instances in these formats with the exception of MPECDUMP.

3.10.1.4.2 ANTIGONE, IPOPT, SHOT, SCIP

• Updated Intel MKL to version 2020.2 on Linux and Mac OS X.

3.10.1.4.3 BARON

• New libraries 20.10.16.
  – Performance improvements including faster automatic differentiation, improved probing, improved local search, and improvements in BARON’s cutting plane routines.
  – Changed default of option WantDual to 1. That is, dual values are now provided by default for nonlinear problems, too.
  – CBC updated to 2.10.5.

3.10.1.4.4 BDMLP

• We plan to drop the LP/MIP solver BDMLP with the next major release. Other free alternatives like Cbc outperform BDMLP.
3.10.1.4.5 BENCH

- We plan to drop the meta-solver BENCH with the next major release. Scripting tools provide much better facilities to generate benchmarks.

3.10.1.4.6 CONOPT

- New library 4.21.
  - Added a warning if the unscaled residuals in a feasible model are large due to very large scale factors.
  - Updated the selection of definitional constraints and improved the numerical behavior.
  - Changed tolerances in the derivative debugger related to very large derivatives.
  - Logic for Definition constraints updated:
    * Infeasibilities are handled with a forward push routine and a backward pull routine.
    * The way dense fill-in is removed is updated and the runtime is improved.
  - Changed a tolerance in the Derivative debugger.
  - Fixed an error in one-dimensional search when there are large 2nd derivatives or discontinuous 1st derivatives. Tried to recover a point that did not exist.
  - Fixed an error with model *hda*: there was a conflict between monotonicity and a tolerance.
  - Fixed some occurrences of system errors 12371, 12372, 88120.
  - Fixed a problem with a very small pivot (non-default tolerance) that created an incorrect penalty constraint.
  - Fixed an error that could appear in models with very long constraints.

3.10.1.4.7 Convert and ConvertD

- Added support for semi-integer and semi-continuous variables and complementarity constraints to scalar JuMP output format.
- Added information about discrete variables to the GDX files created by options Hessian and Jacobian for CONVERTD.
- The Hessian and Jacobian option for CONVERTD report the .stage variable and equation attribute in the variable and equations records only. In the past, the .prior and .scale attributes were reported.
- Fixed an error with suppressed complementarity constraints in Pyomo output.
- Fixed handling of semicontinuous variables with lower bound equal upper bound when writing MPS files.

3.10.1.4.8 Couenne

- As announced, the solver Couenne has been removed from the GAMS system. For the time being, libraries of GAMS/Couenne are available at the COIN-OR GAMSlinks project. Note, that GAMS does not test or offer support for these libraries.
3.10.1.4.9 CPLEX

- The two solver links CPLEX and CPLEXD were merged under the name CPLEX. Features that were formerly available in only one of the two solver links are now all usable with the new CPLEX link. Those features include:
  - Formerly only CPLEX:
    * SOS1 and SOS2 sets
    * Branching priorities
    * Indicator constraints
    * Conflict refiner
    * Ranging / sensitivity analysis
    * Solution pool for mixed-integer (quadratic) problems
  - Formerly only CPLEXD:
    * Better handling of quadratic constraints (e.g. providing duals)
    * Hot start capability in GUSS
    * Solving multiple instances of GAMSModelInstance in parallel using GAMS APIs
    * Solvelink=solvelink.AsyncThreads
    * Cplex remote object and distributed MIP
    * Benders decomposition

- New features:
  - Partial MIP starts: allow to only include variable levels to the MIP start that are integer (within the tolerance tryint, see mipstart)
  - Multiple MIP starts provided via GDX (see multimipstart)
  - Lower / upper bound ranging / sensitivity analysis (see bndrng)
  - Lazy constraints (see userlazyconcall or .lazy)
  - Added MIP statistics (e.g. number of nodes, solve time, objective lower and upper bounds) to BCH call via GDX files usergdxnameinc and usergdxname.

- Added options:
  - Added option multimipstart for multiple MIP starts
  - Added option conflictdisplay
  - Added option multobjtolmip: the default value changes the behavior of the multi-objective optimization for continuous models
  - Added option bndrng: lower / upper bound ranging / sensitivity analysis
  - Added option indicoptstrict: handle erroneous indicator constraints
  - Added option userlazyconcall: adding lazy constraints using BCH
  - Added dot option .lazy: select linear constraints as lazy constraints
  - Added option mipstopexpr: custom MIP stopping criterion to terminate during branch-and-bound

- Changed options:
  - Changed behaviour of rngrestart: If the specified file extension is GDX, the ranging results will be exported as GDX instead of GMS.
  - Changed behaviour of miptracenode and miptracetime: option has no effect and MIP trace is appended when the solver reports global progress
  - Changed behaviour of quality: added entries regarding indicator constraints and MIP condition numbers
  - Changed default of randomseed from 0 to 201909284
3.10 Distribution

- Changed default of `tuningtilim` from 10000 to 1e75
- Changed default of `tuningdettilim` from 1e7 to 1e75
- Changed default of `workmem` from 128 to 2048
- Changed name `rtlcuts` to `rltcuts`
- Added documentation for partial MIP starts to `mipstart`

- Updated documentation for multiobjective optimization about CPLEX's two different strategies for continuous and discrete models (see also `multobjtolmip`).

- Fixed an error in BCH user heuristics where a false objective function of the incumbent was reported to CPLEX (in former CPLEXD link).

3.10.1.4.10 GUROBI

- New libraries 9.1.0.
- New option `rngrestart`: Export sensitivity information as GAMS readable format (GMS or GDX).
- New option `mipstopexpr`: Custom MIP stopping criterion to terminate during branch-and-bound.
- New option `norelheurwork`: controls NoRel heuristic.
- New option `norelheurt ime`: controls NoRel heuristic.
- New option `tunecleanup`: Enables a tuning cleanup phase.
- New option `poolgapabs`: Maximum absolute gap for stored solutions.
- New option `integralityfocus`: Indicator for stricter enforcement of integrality constraints.
- New option `value -1 for option .lazy`: Treat as a user cut.
- Removed option `norelheuristic`. A finer control of Gurobi's NoRelHeuristic is done through the newly introduced options `norelheurwork` and `norelheurt ime`.

3.10.1.4.11 IPOPT

- Changed default of option `acceptable_iter` from 15 to 0. This disables a heuristic in Ipopt to stop early with a solution that doesn't satisfy the usual termination tolerances when progress is slow.

3.10.1.4.12 Kestrel

- Fixed a bug that prevented job submission on Linux and Mac OS X due to a missing certificate.

3.10.1.4.13 Lindo/LindoGlobal

- Added the facility to detect irreducible infeasible and unbounded sets (IIS and IUS) and some options to control this.
- Added support for functions `div`, `log2`, `round`, and `trunc`. 
3.10.1.4.14  LocalSolver

- New libraries 9.5 (20200923).

3.10.1.4.15  LS

- We plan to drop the least-square solver LS with the next major release. The Python method `numpy.linalg.lstsq` included in the GAMS distribution can be used within an embedded Python code section to efficiently minimize the sum of squared residuals.

3.10.1.4.16  MOSEK

- New libraries 9.2.28.

3.10.1.4.17  ODHCPLEX

- More features available in the GAMS link:
  - SOS1 and SOS2 sets
  - Branching priorities
  - Indicator constraints
  - Partial MIP starts: allow to only include variable levels to the MIP start that are integer (within the tolerance tryint, see mipstart).
  - Multiple MIP starts provided via GDX (see multimipstart)

3.10.1.4.18  SCIP

- New libraries 7.0 (5b13bda).
- The workaround to assign a non-continuous variable to a block for problem decomposition (use suffix .prior without priorOpt=1) is not necessary anymore (.stage is now available for non-continuous variables). If one runs a model that continues to use this workaround with this version of GAMS, all non-continuous variables will end up in block 1 independent of the .prior value.

3.10.1.4.19  SHOT

- Gurobi is now available as solver for the MIP or MIQCP relaxation on Mac OS X, too.
- Fixed return of final dual bound found by SHOT to GAMS (objEst attribute).

3.10.1.4.20  SolveEngine

- Dropped link to Satalia SolveEngine because Satalia retired SolveEngine Beta and a successor is not yet available.
3.10.1.4.21 SoPlex

- New libraries 5.0 (2afa64b6).

3.10.1.4.22 XPress

- New libraries 8.10.1.
  - Removed option mislpAlgorithmFinalSLP.
  - Removed option mislpAlgorithmNoFinalRounding.
  - Added option mislpCutStrategy.
  - Added option mipstopexpr: Custom MIP stopping criterion to terminate during branch-and-bound.

3.10.1.5 Tools

3.10.1.5.1 GAMS IDE and GAMS Studio

- Fixed that Bonmin and Ipopt solver option files could not be created or edited via the options editor.

3.10.1.5.2 GAMS Studio

- New version 1.4.2.
  - New feature: Remote execution with GAMS Engine.
  - New feature: Allow to open $[bat|lib]include files by context menu, F2 shortcut, or holding Ctrl and click on the filename.
  - Renamed About GAMS dialog to GAMS Licensing dialog to make it more clear, that this allows to install a GAMS license.
  - Improved GAMS license detection during Studio start and in GAMS Licensing dialog; if the content of a license file is in the clipboard when starting Studio, a dialog to install the license is started.
  - Added support for comma separated list as file filter in search dialog.
  - Changed behavior of ”Whole Words” search option: $ is now considered a word boundary.
  - Reworked NEOS integration, e.g.:
    * Allowing to kill submitted active jobs.
    * Moved NEOS control options into dialog.
    * Changed default parameter from forceWork=1 to previousWork=1.
    * Improved handling of network errors.
  - Minor bug fixes and improvements, e.g.:
    * Added dialog for confidentiality and terms of use confirmation before submitting models to NEOS.
    * Added Arrow Up/Down as shortcuts to jump to search results when Results View is focused.
    * Added tooltips to options in search dialog.
    * Syntax coloring improvements.
    * Fixed file codec selection for the model assembly dialog.
    * Fixed and enhanced reloading mechanism of the GDX Viewer.
    * Fix to ensure that lowercase directory names are always used for MIRO directories, e.g. data\myModel instead of data\myModel.
    * Fixed empty history on welcome page when starting Studio with no tabs open.
    * Fixed potential crash if files are modified in the background while a dialog is open.
3.10.1.5.3 MessageReceiverWindow

- Since GAMS does not support any 32bit platform anymore, this tool is compiled for x64 instead of Any CPU now.

3.10.1.5.4 MODEL2TEX

- Deprecated command line parameter -e has been removed.

3.10.1.6 APIs

3.10.1.6.1 .NET

- Since GAMS does not support any 32bit platform anymore, the .Net DLL is compiled for x64 instead of Any CPU now.

3.10.1.6.2 C++

- Adjusted compilation of examples to new handling of critical sections in expert level APIs.
- Adjusted some examples since GAMS/CplexD is now a synonym for GAMS/Cplex.

3.10.1.6.3 Python

- New experimental API gams_magic that enables the use of GAMS in Jupyter notebooks has been added. This release of gams_magic should be considered a beta product and will be subject to changes in the future that may result in compatibility issues.
- New API gams_engine that allows the creation of Python clients for GAMS Engine has been added.

3.10.1.7 Expert Level APIs

- The new files gcmt.c and gcmt.h are now required when compiling/linking C expert level API files (e.g. gdxcc.c/h) to protect critical sections in the interface for multi-threaded applications. On some platforms, this might also require linking against additional libraries, e.g. libpthread on Linux. Adding the define GC_NO_MUTEX when compiling the C expert level API files provides the old behavior.
- Added new call gdxStoreDomainSets to the GDX API to control the use of one-dimensional sets as domain sets. Disabling the storage of domain information for a one-dimensional set when writing to GDX saves memory but prevents the set from being used for domain checking.

3.10.1.8 Model Libraries

3.10.1.8.1 GAMS API Library  New models:

- Pgams_engine.gms : Test gams_engine API (60)
- Pneos.gms : Test submitting GAMS models to NEOS server (61)
### 3.10.1.8.2 GAMS Test Library

**New models:**

- *lindgl05.gms* : Test new functions added to GAMS/Lindo(Global) (828)
- *cplex04.gms* : CPLEX test suite - free gams model test (829)
- *cplex05.gms* : CPLEX test suite - interactive (830)
- *cplex06.gms* : CPLEX test suite - multiobjective (831)
- *cplex07.gms* : CPLEX test suite - bch incumbent checker (832)
- *sens01.gms* : sensitivity / ranging test (833)
- *mipstophexpr1.gms* : mip stop expression test (834)
- *attrib01.gms* : Test attribute setting and preservation in Convert (835)
- *gamsjupyter01.gms* : Test GAMS Jupyter Notebooks (836)
- *cmi01.gms* : Test captureModelInstance option with all solveLink combinations (837)

### 3.10.1.9 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAEC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.13</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLNP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.10.2 33.2.0 Minor release (December 01, 2020)

#### 3.10.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release.

#### 3.10.2.2 GAMS System

##### 3.10.2.2.1 GAMS

- Fixed some problems with Implicit set definition:
  - When doing a filtered load from GDX or embeddedCode, the defined set only gets the elements actually loaded but not the ones that were filtered out.
  - Merging to start of existing data (with `$onMulti`) does not lead to out-of-order data anymore.

#### 3.10.2.3 Solvers

##### 3.10.2.3.1 CPLEX

- Fixed issue where option writemst did not produce any output.

##### 3.10.2.3.2 KNITRO

- Add workaround to avoid abnormal termination of Knitro 12.2 libraries on AMD processors.

#### 3.10.2.4 Tools

##### 3.10.2.4.1 GAMS Studio

- New version 1.4.5 with various bug fixes, stability improvements and minor enhancements, e.g.:
  - Added error output to system log if the given MIRO installation location is invalid.
  - Disabled run actions and MIRO menu when active group has no active executable file.
  - Fixed a problem with GAMS parameters being ignored in rare cases.
  - Fixed a potential crash when reloading a large `gms` file while the syntax highlighter was still processing that file.
  - Improved Search and Replace, e.g.:
    * Improved performance of Find Next/Previous.
    * Added possibility to interrupt Find Next/Previous.
    * Fixed a problem which did not allow to jump to search results beyond the 50,000th match of the search term.
    * Fixed handling of wrapped lines in search.
  - Improved MIRO and Engine dialogs, e.g.:
    * Removed standalone MIRO assembly file dialog and added it to a new reworked MIRO deploy dialog instead.
    * Fixed version comparison in GAMS Engine dialog.
3.11 32 Distribution

3.11.1 32.1.0 Major release (July 31, 2020)

3.11.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Youngdae Kim, Michaja Pehl, Thomas Rutherford, and Igor Sheleg.

3.11.1.2 Platforms

- Dropped support for Sparc Solaris and AIX systems.

3.11.1.3 GAMS System

3.11.1.3.1 GAMS

- New default values for some GAMS options:

<table>
<thead>
<tr>
<th>Option</th>
<th>new default</th>
<th>old default</th>
</tr>
</thead>
<tbody>
<tr>
<td>logOption</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>empty</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>errMsg</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>errorLog</td>
<td>2147483647</td>
<td>0</td>
</tr>
<tr>
<td>lstTitleLeftAligned</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>putPW</td>
<td>32767</td>
<td>255</td>
</tr>
<tr>
<td>pageContr</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>pageSize</td>
<td>0</td>
<td>58</td>
</tr>
<tr>
<td>pageWidth</td>
<td>32767</td>
<td>255</td>
</tr>
<tr>
<td>optCR</td>
<td>0.0001</td>
<td>0.1</td>
</tr>
<tr>
<td>intVarUp</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>resLim</td>
<td>1e10</td>
<td>1000</td>
</tr>
<tr>
<td>solveLink</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>iterLim</td>
<td>2147483647</td>
<td>2e9</td>
</tr>
</tbody>
</table>

In order to work with the default values active in GAMS 31, copy the file `gams31config.yaml` (part of the GAMS system directory) into the location searched by GAMS for configuration files (e.g. the GAMS system directory) with the name `gamsconfig.yaml`.

- Extended the impact of command line parameter `etLim`, so that it reduces `resLim` for models solved automatically if that is greater than `etLim - timeElapsed`.

- Added new option `zeroToEps` to allow reading zero values as EPS at execution time.

- Added new options `procTreeMemMonitor` and `procTreeMemTicks` to enable monitoring of a memory high-water mark for the GAMS process and all its children.

- Adjusted behavior of command line parameters `restart`, `restartNamed`, `save`, `saveObfuscate`, `xsave`, and `xsaveObfuscate`: If a file name is given including a file extension, this extension is kept. In the
past it was replaced by a default extension. Also, '?' is not allowed in the file name when writing a save file anymore.

- Extended influence of command line parameter inputDir, so that it also extends the search path for GDX files loaded with $gdxIn.
- Extended the search space for files included via $libInclude, extrinsic function libraries, and external equation libraries: Now, the directories inclib, extrinsic_functions, and external_equations, resp., are checked in all GAMS standard locations before [GAMS System Directory]/inclib, [GAMS System Directory], and [GAMS Working Directory], respectively.
- Renamed the command line parameter execErr to maxExecError to be in sync with the related function maxExecError. The old name is still available as a synonym.
- Added command line parameter previousWork to instruct GAMS to write a workfile using the format of an older GAMS version. This is useful in particular with submission to the NEOS server.
- Adjusted the function values returned by the power functions (e.g. $x^{**}y$). In some cases where $x$ is very small and positive the derivatives may become unbounded but the function values are well-behaved and can still be computed precisely and without overflow.
- Fixed a problem on Linux when the XGD_CONFIG_DIRS or XGD_DATA_DIRS variables have more than 7 directory entries.

3.11.1.3.2 GMSPython

- Added the packages pandas, numpy, openpyxl, schema, tqdm and their dependencies.

3.11.1.3.3 Mac OS X Installer

- New way to install GAMS on Mac OS X using a PKG installer, see the installation notes.

3.11.1.3.4 Windows Installer

- Added a checkbox for enabling/disabling the creation of a start menu entry to the Select Start Menu Folder Wizard page. The default state of the checkbox is inherited from previous installations.
- Command line argument /noIcons can be used in order to check the Don’t create a Start Menu folder checkbox on the Select Start Menu Folder page.
- New command line argument /desktopIcons=yes|no (default:yes) can be used in order to turn on/off the creation of Desktop icons.
- The installer now writes all registry entries either to HKEY_CURRENT_USER\Software\Classes (current user) or HKEY_LOCAL_MACHINE\Software\Classes (all users).
- Show warning in case the system GAMSDIR environment variable is updated, but a user GAMSDIR environment variable exists and not vice versa.
- Always notify the operating system about changed environment variables.
- Changed the default installation location from C:\GAMS\win64\XX.Y to C:\GAMS\XX (e.g. C:\GAMS\32).
- Desktop shortcut names have been changed to GAMS IDE 32 and GAMS Studio 32.
- Start menu entry reads GAMS XX (e.g. GAMS 32) instead of GAMS XX.Y (win32/win64).
- Fixed a bug that prevented file associations (.gdx and .gms files) from working properly in certain cases.
3.11.1.4 Solvers

3.11.1.4.1 Bonmin, CBC, Couenne, LocalSolver, Osi links, SHOT, SoPlex

- Fixed that a setting of GAMS option `iterlim` to 2000000000 (the previous default) was not passed on to the solver.

3.11.1.4.2 CONOPT

- Added Conopt4 to the thread-safe solvers. It now can be used with `solveLink=6`.

3.11.1.4.3 CONVERT

- New output format: JuMP scalar model.

3.11.1.4.4 GUROBI

- Fixed a bug introduced with 31.1.0 and only partly fixed in 31.1.1 where the `GRB_LICENSE_FILE` environment variable was incorrectly dropped.

3.11.1.4.5 Ipopt

- Updated Mumps to 5.3.3.

3.11.1.4.6 KESTREL

- Kestrel interface source code has been ported from Python 2 to Python 3.
- Also on Windows, the source code of the link is now distributed instead of a binary.
- The GAMS distributed Python interpreter (`GMSPython`) is used for running the Kestrel client.

3.11.1.4.7 KNITRO

- New libraries 12.2.2.
  - Knitro 12 adds many improvements to the Knitro mixed-integer (MIP/MINLP) branch-and-cut solver. Several new cuts controlled by Knitro user options have been introduced including mixed-integer rounding cuts, zero-half cuts and clique cuts. In addition, improvements were made to the knapsack cuts implementation.
  - Knitro 12 offers several enhancements to the Knitro presolver. The presolver has been extended to allow multiple passes through the presolve operations. There are new presolve operations for variables in complementarity constraints and to tighten variable bounds.
  - Knitro 12 offers efficiency and storage/memory improvements in the callable library C API - the one used in the GAMS link!
  - Knitro 12.2 enables default parallelism when using multi-start or multi-algorithm features.
  - Knitro 12.2 offers several performance speed-ups:
    * on very large models with the default (i.e. interior-point) algorithm: potentially dramatic
    * when using the limited-memory BFGS Hessian option
    * in the SQP algorithm
  - A number of minor bug fixes and performance improvements are included in this release: rare segmentation fault with knapsack cuts fixed, bug in convexity detection fixed, poor/slow choice of algorithm for LP sub-problems fixed, poor/slow presolve issue on MIP models fixed.
3.11.1.4.8 LocalSolver

- New libraries 9.5 (20200701).
- Setting iterlim to the maximal possible value (2147483647) will not change the default iteration limit of LocalSolver anymore, see also the documentation.

3.11.1.4.9 Mosek


3.11.1.4.10 ODHCPLEX

- New libraries 5.2.2.
- New options integerTol, subOrder, subPresolve, zeroTol, penPerturb, and threadZeroSync have been added. The options newCallback and variableClean allow for more option values.
- New handling of solutions that are not quite integer but within the integer feasibility tolerance of CPLEX but become infeasible if one rounds the integer variables. ODHCPLEX used to reject such solutions (while in CPLEX (and other MIP solvers) such solutions are considered acceptable for the user).

3.11.1.4.11 SoPlex

- Setting iterlim to the maximal possible value (2147483647) will not change the iteration limit of SoPlex anymore, so that SoPlex will run without a limit.

3.11.1.4.12 XPRESS

- New libraries 8.8.5.
- New solver: XPRESS SLP and XPRESS Knitro for nonlinear (mixed-integer) programming.
  - XPRESS SLP is a sequential linear programming (SLP) solver and XPRESS Knitro equals Knitro, but the problem is passed through the XPRESS nonlinear presolve.
  - Both can solve convex MINLP to global optimality and act as a heuristic method for general nonconvex problems.
  - XPRESS SLP offers a multistart to run different configurations in parallel or to start with multiple initial guesses.
  - If both XPRESS SLP and XPRESS Knitro are licensed, XPRESS can choose the nonlinear solver automatically based on instance attributes.
  - For more information about the XPRESS SLP algorithm, see FICO Xpress Optimization Help (Nonlinear Reference Manual).
  - XPRESS SLP and XPRESS Knitro are available on all supported platforms.
  - XPRESS SLP requires a license for continuous nonlinear models or for mixed-integer nonlinear models.
  - XPRESS Knitro requires a GAMS/Knitro license in addition to the XPRESS license for (mixed-integer) nonlinear models.
3.11.5 Tools

3.11.5.1 CHK4UPD

- If the components of the checked license have different expiration dates, the validity of the license always refers to the base component now. In the past, the component with the oldest expiration date was used.

3.11.5.2 FINDTHISGAMS

- Support for registry key changes of the Windows installer has been added.
- New command line parameters list, delete, write, and ide. See the documentation for detailed information.

3.11.5.3 GAMS Studio

- New version 1.3.3.
  - New feature: Code Folding
    * Shortcut Alt+L switches folding state of the current block.
    * Shortcut Alt+0 folds and Shift+Alt+0 unfolds all foldable blocks.
    * Code folding is possible for pairs of parentheses that span multiple lines or Dollar Control Options that come in pairs.
  - New feature: Remote execution on NEOS Server for Optimization
    * Models can be executed remotely via the menu GAMS -> Run NEOS or the execute button located in the toolbar.
    * Note: Usually it takes some time before the GAMS version at NEOS gets updated. During that time one might get an error because of an incompatible workfile version. This can be avoided by setting the command line parameter previousWork to 1.
  - New feature: Navigation History
    * Mouse button 4/5 or Alt+Left/Right on Windows/Linux and Ctrl+[/] on Mac OS X can now be used to jump back to a previous text cursor positions or forward again. This is helpful to quickly navigate between different locations in files that one is currently working with.
  - New feature: Distraction Free Mode
    * Distraction Free Mode quickly turns off everything except for the editor window for better focus on modeling work. This is especially useful for small screens. When leaving Distraction Free Mode all Studio widgets will be put back where they were. To switch to and from Distraction Free Mode go to the Menu > View > Distraction Free Mode, or press Ctrl+Alt+Enter on Windows and Linux, or Cmd+Option+Enter on Mac OS X.
  - New feature: Print support for text files (gms, txt, lst, dmp, ...). This is accessible with the shortcut Ctrl+P or in the menu File -> Print.
  - Added automatic switch to first non-empty tab when searching the Model Library Explorer. The old tab selection is restored if that tab contains results again.
  - Added header labels to copy actions in GDX Viewer table view. Two additional entries that allow to copy without the labels are available via the context menu.
  - Added dmp extension to list of executable file types.
  - Added Shift+Option+Up/Down hotkey to start block-edit on Mac OS X.
  - Changed terminal to native Mac OS X Terminal on Mac OS X.
Stability improvements and minor bug fixes, e.g.:

- Fixed syntax highlighter not recognizing "+" and "+" in "Model" statement.
- Fixed missing LXI pane for files containing more than one dot.
- Fixed group name for gms files containing more than one dot in the name.
- Fixed rare random crash when opening the MIRO Model Assembly File dialog.
- Fixed some icons not following selected theme.
- Fixed line wrapping behavior not following settings.
- Fixed that bookmarks in help view were not being saved permanently.
- Fixed crash on reading LXI files containing empty lines.
- Fixed some issues with the license file detection and info in the "About GAMS" dialog.
- Fixed missing reload if a opened GDX file gets recreated.

3.11.1.5.4 MODEL2TEX

- Source code of model2tex has been ported to Python 3 and does not support Python 2 anymore.
- Distribute source code instead of binary on Windows. model2tex.exe has been replaced by model2tex.cmd.
- The GAMS distributed Python interpreter (GMSPython) is used for executing model2tex.
- Command line parameter -e has been deprecated and will be removed in the future.

3.11.1.6 APIs

- Automatically finding a GAMS installation from the Windows registry has been changed according to the changes to the Windows installer ([HKEY_CURRENT_USER\HKEY_LOCAL_MACHINE]\Software\Classes\gams.location).
- The performance of the Object-oriented APIs (in particular the class GAMSDatabase) has been improved.

3.11.1.6.1 .NET

- Change to GAMSWorkspace.AddCheckpoint: The checkpointName is now determined automatically as well when given as string.Empty, not only when set to null or being omitted.

3.11.1.6.2 Python

- As announced, dropped support for Python 2. The Object-oriented API as well as the expert-level APIs can be used with Python 3.6, Python 3.7, and Python 3.8.
- An experimental object oriented data interface called gamstransfer has been added. This interface is built on the expert-level Python APIs and simplifies the work of reading and writing GDX files. The underlying data structure is built on structured numpy arrays; this structure allows data to be easily converted directly to Pandas DataFrames (or other native Python data types). This release of gamstransfer should be considered a beta product and will be subject to changes in the future that may result in compatibility issues.
- New experimental API gams2numpy for communicating data between GAMS and numpy arrays has been added. This release of gams2numpy should be considered a beta product and will be subject to changes in the future that may result in compatibility issues.
3.11.1.7 Model Libraries

3.11.1.7.1 GAMS Model Library  New models:

- scenmerge.gms : Combining scenario results in a directory tree with gdxmerge (427)

3.11.1.7.2 GAMS Test Library  New models:

- zerotoeeps1.gms : Test data loading with $onEps and option zeroToEps (821)
- idir01.gms : Test including files from different locations (822)
- mrw02.gms : Test MessageReceiverWindow.exe from Python (823)
- memmon1.gms : Test procTreeMemMonitor=1 behavior (824)
- gams2numpy01.gms : Test gams2numpy Python API string mode (826)
- gams2numpy02.gms : Test gams2numpy Python API raw mode (827)

3.11.1.7.3 GAMS API Library  New models:

- Pgamstransfer.gms : Test gamstransfer API (58)
- Pgams2numpy.gms : Test gams2numpy API (59)

3.11.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHEEC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.13</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>x86 64bit MS Windows</td>
<td>x86 64bit Linux</td>
<td>x86 64bit Mac OS X</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 35.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3.11.2 32.2.0 Minor release (August 26, 2020)

3.11.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Charles Chan, Ricardo Lima, Sandro Konrad Luh, Ami Reznik, Tom Rutherford, and Berk Uzundere.

3.11.2.2 GAMS System

3.11.2.2.1 GAMS

- Introduced new values 3 and 4 for option savePoint that allow to save the point file in the scratch directory.
- Fixed a bug which in rare cases caused GAMS to write files exceeding 80,000 characters per line (and thus the maximum line length GAMS can handle) when the command line parameter dumpopt was set.
- Fixed a bug in restart files that incorrectly stored the location of the GAMS configuration file (gmscmpXX.txt).
- Fixed a bug that is related to providing the GAMS configuration file (gmscmpXX.txt) explicitly via command line parameter subSys.
- Fixed reporting of compilation errors in encrypted input files.
- Fixed the count of discrete variables in the model statistics in case any discrete variable was fixed.

3.11.2.2.2 Windows Installer

- Fixed a bug that prevented registry entries from being removed correctly during uninstallation.
3.11.2.3 Solvers

3.11.2.3.1 Gurobi

- New libraries 9.0.3.

3.11.2.3.2 Lindo/LindoGlobal

- New libraries 12.0.208.
  - Fixed a rare problem with stochastic problems.

3.11.2.3.3 Mosek

- New libraries 9.2.18.

3.11.2.3.4 SCIP

- New libraries 7.0 (80549b8905).
- PaPILO updated to 1.0 (d0e5444cd).

3.11.2.3.5 SoPlex

- New libraries 5.0 (3623dbc0).

3.11.2.4 Tools

3.11.2.4.1 GAMS Studio

- New version 1.3.4.
  - Minor bug fixes, e.g.:
    * Show community license type in the solver matrix.
    * Fixed missing tab browser button when Studio starts in maximized or full-screen mode.
    * Fixed insertion of auto-closing characters in unwanted situations (e.g. beginning of a word).

3.11.2.4.2 GDX2VEDA

- Fixed a bug which in rare cases crashed the tool for large data sets on Windows.

3.11.2.5 Model Libraries

3.11.2.5.1 GAMS Test Library

- Updated models:
  * savep1.gms: execute_loadpoint: save point - clear - loadpoint - reoptimize (65)
3.12 31 Distribution

3.12.1 31.1.0 Major release (May 01, 2020)

3.12.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Christoph Böhringer, Wolfgang Britz, Andreas Lundell, Scott McDonald, Victor Nechifor, Renger van Nieuwkoop, A. Omidi, Christoph Pahmeyer, Soummya Roy, Thomas Rutherford, Alireza Soroudi, Tim Varelmann, and Elizabeth Wong.

3.12.1.2 Platforms

- As announced, dropped support for 32-bit MS Windows.

3.12.1.3 GAMS System

3.12.1.3.1 GAMS

- Extended the syntax of the model statement to allow to restrict the domain of variables used in the model at one spot.
- Added dollar control options $on/offSuffixDLVars and command line parameter SuffixDLVars to control whether it is allowed to use suffixes on variables with limited domains in a model.
- Added new command line parameters to initialize the state of certain dollar control options: Digit, Empty, EolCom, Filtered, InlineCom, Listing.
- Extended the behavior of the command line parameter checkErrorLevel, so that it also initializes the state of the dollar control option $on/offCheckErrorLevel.
- Extended the behavior of the command line parameter strictSingleton, so that it also initializes the state of the dollar control option $on/offStrictSingleton.
- Added new command line parameters to set the default values for certain put file attributes: PutND, PutNR, PutPS, PutPW.
- Removed the limit on the number of significant digits allowed in floating-point values in GAMS source.

Historically, GAMS has run on widely different platforms where even the floating-point arithmetic varied. To ensure the precision specified in the GAMS input didn't exceed what could be stored and computed with as GAMS ran, we limited the amount of precision (i.e. the number of digits used for numbers in the GAMS source) that could be specified. This limit could be relaxed by using the $offDigit control.

Modern computers all support IEEE double-precision arithmetic. It's a well-accepted and familiar standard, and the new behavior - to convert the decimal value in the GAMS source into the nearest double-precision value, regardless of the number of digits specified - is what is generally expected. There is a limit (currently 40) on the number of digits treated as significant: digits beyond this point are quietly treated as zero.

For "nice" values (i.e. those with fewer than 17 digits of precision and exponents of limited size, like 3.14159 or 2.5032e10) there is no difference in the old and new behavior. But for values specified with 17 or more digits of precision or having exponents of large size, the double-precision values can vary slightly between the old and new systems. The new values are the correct ones.

The new command line parameter sys18 has been added in case any users require the old behavior.
• New options `encryptKey` and `decryptKey` to encrypt and decrypt source based on a key rather than a target license. This allows developers to create encrypted input files that can be executed with any GAMS license and the right key.

• GAMS will search standard locations for the license file `gamslice.txt` including the GAMS system directory. The standard locations for your operating system are described in the installation notes. You can also continue to use the `license` command line parameter to point to a specific license file.

• GAMS can now use a configuration file `gamsconfig.yaml` to specify default values for command line parameters and environment variables: see details in GAMS Configuration in YAML Format. This can be used to implement user-specific default option settings that are shared between different GAMS versions. In the future, we also expect to use this capability to support the setting of options and defaults that support backward compatibility; this will become important as we change GAMS defaults in significant ways to modernize the user experience in the light of current computing environments and user expectations.

• Added system suffixes `%system.licenseFileName%`, `%system.userConfigDir%`, and `%system.userDataDir%`.

• Embedded Python Code Facility:
  - Embedded Python code can be used with Python 3.6, 3.7, and 3.8. GMSPython is used for embedded code by default on the core platforms.
  - The GAMS command line parameter `pySetup` has been deprecated. The environment variable `GMSPYTHONLIB` that needs to point to the Python library (e.g. `python38.dll` on Windows) determines the Python installation to be used in embedded Python code.
  - The `gams.set` method to write data back to GAMS has a new argument `mapKeys` that allows to pass a callable to remap the elements of the key.

• Removed GAMS return code 116 ("Could not find GMSPython").

• Fixed the `card` operator for scalar symbols, so that it always returns 1 when the symbol has data. In the past it might have been wrong for symbols that had no data in the declaration, but just from an assignment.

• Fixed issues where GAMS would crash on exceptionally large models (more than 268 million rows or columns) with certain non-default options selected.

• Fixed a problem which allowed multiple declarations of the same equation or variable when `$offMulti` was active.

• Fixed some 3-argument intrinsic functions whose third argument was optional (e.g. `ncpVUSin`) - these were not taking the default value for the third argument correctly into account when called with two arguments.

• Fixed a bug which caused wrong execution of `$onPut` and `embeddedCode` after Programming Flow Controls that were not terminated with a semicolon.

• Fixed a bug which caused variable and equation records fixed at 0 to be dumped incorrectly with `dumpOpt`.

• Fixed incomplete warning when writing a non-default status of `$on/offEmpty` to a save file.

• `execMode` did not prevent the execution of `scriptExit`. This has been fixed.

3.12.1.3.2 GMSPython

• GMSPython has been upgraded from Python 3.6 to Python 3.8 and comes without Python package installer `pip`.

• Added the package `ply`. 
3.12.1.3.3 Libinclude

- The libinclude file `rank.gms`, which is based on the `gdxrank` tool, got a complete overhaul. The temporary GDX files resides now in the scratch directory, so multiple GAMS models using `rank.gms` can coexist in the same working directory. Moreover, the percentile calculation works with sort values that may include +inf/-inf. Some checks have been added that result in execution errors in case the operation was not successful, e.g. if `gdxrank` failed to sort.
- Fixed a bug in `pyEmbMI.gms` that prevented string options from being set properly.

3.12.1.3.4 Documentation

- Improved the layout. The new layout provides a more responsive view on a smaller screen.

3.12.1.3.5 Windows Installer

- Default installation mode has been changed from `All Users` to `Current User`. This can be changed on the Advanced Options page or by providing `/allUsers=yes` when running the installer from a command prompt.
- The default application used for associating GAMS (.gms) and GDX (.gdx) files has been changed from the GAMS IDE to GAMS Studio.
- New command line argument `/allUsers=yes|no` (default: no) can be used in order to install for `All Users` instead of `Current User`.
- New command line argument `/noLicense=yes|no` (default: no) can be used in order to prevent any GAMS license from being written.
- New GAMS License page:
  - Allows to select a GAMS license to be used. The installer tries to find GAMS licenses in different locations and automatically selects a license to be used with the following precedence:
    * Clipboard
    * C:\Users\username\Documents\GAMS
    * Previous GAMS installation found from the registry (User)
    * Previous GAMS installation found from the registry (System)
  - A license file is written to C:\Users\username\Documents\GAMS when installing for Current User (default). Installing for All Users or checking Write License to System Directory will write the license to the system directory instead.

3.12.1.4 Solvers

3.12.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2020.0 on Windows.

3.12.1.4.2 BARON

  - Improved local search strategies.
  - Increased effectiveness of cut pool manager.
  - Improved relaxations for MIQPs.
  - Added new LP subsolver based on HSL’s LA04.
3.12.1.4.3 CBC

- GAMS/CBC did not use the default value for all parameters as documented and as used in the CBC standalone program. This has been fixed. To restore some of the previous behavior, the following parameter setting can be used:

```plaintext
nodeStrategy hybrid
preprocess off
cuts off
cliqueCuts off
flowCoverCuts off
gomoryCuts off
knapackCuts off
mirCuts off
twoMirCuts off
probingCuts off
zeroHalfCuts off
heuristics 0
feaspump off
greedyHeuristic off
roundingHeuristic off
```

3.12.1.4.4 CONOPT

- New libraries 3.13L.
  - Added new facilities to print the variables changed by the pre-processor. See options prprec, rtpprec, and rtprel.

- New libraries 4.19.
  - Some errors related to the duals and basic/nonbasic flags for some definitional constraints and forcing constraints have been corrected.
  - The two tolerances Tol_IFixed and Tol_DFixed for when a variables is fixed have been unified into a single tolerance, Tol_Fixed.
  - The routine for detecting definitional constraints has been simplified and some options from previous versions are no longer used (Flg_DC_Uinique, Lim_DFVars, and Tol_Def_Ini) and Tol_Def_Mult has a new default value.

3.12.1.4.5 CONVERT

- Added support for option HeaderTimeStamp, which was added for ConvertD before, in Convert.

3.12.1.4.6 COUENNE

- We plan to drop Couenne with one of the next major releases.

3.12.1.4.7 CPLEX

- Fixed incorrect display of selection of LPMethod when rerunning without presolve for infeasible model.
- Fixed incorrect selection of LPMethod when rerunning without presolve for infeasible model in CplexD.
3.12.1.4.8 GUROBI

- New libraries 9.0.2.
- If GAMS/Gurobi is licensed, the setting of the environment variable `GRB_LICENSE_FILE` is ignored. The setting of this environment variable interfered with the GAMS/Gurobi licensing.

3.12.1.4.9 IPOPT

- New libraries 3.13.2.
  - MA28 can no longer be loaded at run-time from a user-provided HSL library. Removed option 'ma28_pivot'.
  - The default of option `ma77_order` has changed from 'amd' to 'metis'.
  - The default of option `ma97_print_level` has changed from 0 to -1.
  - The default of option `mumps_pivot_order` has been changed from 7 to 5 to work around a bug in MUMPS.
- Updated MUMPS to version 5.3.1. MUMPS has been build with OpenMP support enabled.
- Updated METIS (used by HSL codes and MUMPS) to version 5.1.0.

3.12.1.4.10 KESTREL

- Fixed a problem with the control file version that prevented Kestrel from being used in GAMS 30.

3.12.1.4.11 Lindo/LindoGlobal

- Increased the number of allowed nonlinear nonzeros in the global solver from 30 to 50 when operating under a demo license.

3.12.1.4.12 LocalSolver

- New libraries 9.5 (20200409).
  - Performance improvements on mixed-variable models like unit commitment or network design problems.
  - Performance improvements on routing and scheduling problems like CVRPTW or jobshop scheduling problems.
  - Better and faster lower bounds for nonlinear problems as well as for combinatorial problems like TSP.
- New parameter `verbosity`.
- Fixed handling of semi-integer variables.

3.12.1.4.13 MOSEK

- New libraries 9.2.4.
- Avoid MOSEK error 1243 on general nonlinear models.
3.12.1.4.14 SCIP

- New libraries 7.0.0 (0bc4dc9c65).
  - The symmetry code has been completely restructured and modularity has been increased.
  - It is now possible to use orbitopes (i.e., polyhedral symmetry handling techniques) and
    orbital fixing on the same instance.
  - Branching decisions can now take symmetry information into account by filtering
    first variables from orbits (new parameter `branching/relpscost/filtercandssym`) and
    transferring pseudo cost information to variables in orbit (new parameter
    `branching/relpscost/transsympscost`).
  - Improved performance of orbital fixing and orbitope propagation.
  - Branching decisions can now take symmetry information into account by filter-
    ing first variables from orbits (new parameter `branching/relpscost/filtercandssym`) and
    transferring pseudo cost information to variables in orbit (new parameter
    `branching/relpscost/transsympscost`).
  - Improved detection of packing/partitioning orbitopes.
  - New possible value 3 for `misc/usesymmetry` and changed default from 2 to 3.
  - Renamed parameter `propagating/orbitalfixing/enableafterrestart` to `propagating/symmetry/recomputeafterrestart`
    which decides whether to recomputate symmetries after a restart or not.
  - Changed default of `constraints/orbite/propfreq` from 5 to 1.
  - Changed default of `constraints/orbite/separafreq` from 5 to -1.
  - Removed parameters `constraints/orbisack/checkalwaysfeas`, `constraints/orbitope/checkalwaysfeas`, `constraints/symresack/checkalwaysfeas`.
  - Removed parameters `constraints/orbisack/checkalwaysfeas`, `constraints/orbitope/checkalwaysfeas`.
  - Removed parameters `propagating/orbitalfixing/symcomptiming`, `propagating/orbitalfixing/performpresolving`, and `propagating/orbitalfixing/recomputerestart`.
  - New parameter `constraints/orbite/usedynamicprop` to enable propagation of or-
    bitope constraints by reordering the rows based on the branching strategy.
  - New parameter `constraints/symresack/checkmonotonicity` to enable upgrade to pack-
    ing/partitioning symresacks even if the underlying permutation is not monotone.

- Presolve:
  - The presolve library PaPIL0 (Parallel Presolve for Integer and Linear Optimization) 1.0
    has been integrated as additional presolver for mixed-integer linear programs.
  - Expressions of form $|x|^p x$ in a nonlinear constraint are now sometimes recognized and
    handled by abspower constraints.
  - Some more quadratic constraints are now recognized as second-order cone constraints.
  - New presolver dualsparsify that tries to combine variables to cancel nonzero coefficients in
    the constraint matrix of a MIP.
  - Improved performance of tworowbnd presolver. New parameters added. Changed default of
    `presolving/tworowbnd/priority` from -500000 to -2000.
  - Extended presolver duallinear by the ability to perform convex combinations of continuous
    variables to calculate better bounds for the dual variables. New parameters added.
  - Huge clique tables are now avoided by restricting the number of nonzero entries relative to
    the number of nonzeros in the problem. New parameter `presolving/clqtablefac`.
  - New parameter `constraints/linear/extractcliques` to turn clique extraction off.
  - Improved presolver "domcol" to not require a complete representation of all constraints in
    the matrix.
  - Enabling aggressive presolving now activates all available presolving plugins, and decreases
    parameter `presolving/restartfac` correctly with respect to default.
* New parameter constraints/linear/maxdualmultaggr and constraints/linear/maxmultaggr to limit the maximum coefficient dynamism of an equation on which multiaggregation is performed. Smaller values make multiaggregations numerically more stable.

* Changed default of constraints/quadratic/empathy4and from 0 to 2. This leads to stronger but larger reformulations for products of binary variables.

− Primal heuristics:
  * New Large Neighborhood Search heuristic "Trust region", available as both a standalone heuristic and a neighborhood inside of Adaptive Large Neighborhood Search.
  * New heuristic Adaptive Diving, which registers all publicly available dive sets from other diving heuristics, learns online which divesets reach the best score, and executes them more frequently.
  * New penalty alternating direction heuristic PADM that splits the problem into several sub-SCIPs according to a user-provided decomposition (see below). The sub-SCIPs are solved on an alternating basis until a feasible solution of the original problem is found.
  * The GINS heuristic can make use of a user-provided decomposition (see below) by selecting a block of variables that maximizes the potential, and randomly selecting a start variable for the neighborhood and/or by selecting an interval of consecutive blocks as neighborhood, until fixing rate is reached. In this case, no variable is randomly selected. Several new parameters added for the use of a decomposition in GINS.
  * The LP face heuristic now wastes less time if it decides not to run.
  * Improved performance of Adaptive Large Neighborhood Search heuristic on merely continuous problems.
  * All primal heuristics that use sub-SCIPs are now disabled within the heuristics fast emphasis setting.
  * Replaced parameter heuristics/localbranching/useuct by heuristics/useuctsubscip, which affects all LNS heuristics.
  * New parameters heuristics/shiftandpropagate/minfixingratelp and heuristics/locks/minfixingratelp to stop the heuristics after propagating integer fixings if no sufficient fixing of the all variables (including continuous) could be achieved.
  * Changed default value of heuristics/coefdiving/freq from 10 to -1.
  * Changed default value of heuristics/conflictdiving/freq from -1 to 10.
  * Changed default value of heuristics/conflictdiving/lockweight from 1.0 to 0.75.
  * Changed default value of heuristics/rins/nodes from 0.1 to 0.3 (due to other code changes, this should not affect default behavior).
  * New value 1 available for heuristics/gins/potential to allow computation based on local LP solution.
  * The display column for memory usage ("mem") now shows the creator name for every new incumbent solution. The heuristic display characters now represents the type of the heuristic (diving, Large Neighborhood Search, propagation, etc.). Changed default of display/width from 139 to 143.

− Branching:
  * New rule vanillafullstrong, mostly for scientific purpose.
  * Improved branching point selection when branching on nonlinear variables. Instead of using exactly the LP solution, a point closer to the middle of the variables domain is chosen. Added parameters branching/midpull and branching/midpullreldomtrig.
  * Many updates to parameters of lookahead branching rule.
  * New parameter branching/relpscost/degeneracyaware to switch degeneracy-aware hybrid branching, which adjusts weights of different scores in relpscost (hybrid) branching rule based on degeneracy information and skips strong branching for very high degeneracy rates.

− Separation:
  * Improvements in cut presolving and cgmip separator.
  * New parameter separating/cgmip/genprimalsols that allows to generate initial primal solutions from Gomory cuts.
* Changed default of `separating/rapidlearning/freq` from -1 to 5. Various new parameters for `rapidlearning`.
* Changed default of `separating/efficacyfac` from 1.0 to 0.6.

- Approximations on the search tree completion and estimates on the tree size have been added.
  * The approximate search tree completion is shown in a new display column.
  * The tree size estimate is used to trigger a restart if, after a reasonable initialization, the estimated size of the remaining tree is large.
  * New parameter group `branching/treemodel` to specify the tree model.

- SCIP can now store user decompositions. A GAMS user can specify one decomposition via the `.stage` variable suffix. Statistics regarding the decomposition are printed to the log before presolving (if `display/statistics` is enabled) and after presolving. New parameter `decomposition/maxgraphedge`.

- Extended the dual proof analysis for infeasible LPs to consider also locally valid rows (new parameter `conflict/uselocalrows`).

- If a reference value is given, the primal-reference and reference-dual integrals are calculated automatically and printed within the SCIP statistics.

- Emphasis settings for numerically challenging instances have been added. They increase numerical stability of (mostly) presolving operations such as (multi-)aggregations at the cost of performance.

- Renamed parameters `misc/allowdualreds` and `misc/allowobjprop` to `misc/allowstrongdualreds` and `misc/allowweakdualreds`, respectively.

- Replaced parameter `numerics/lpfeastol` by `numerics/lpfeastolfactor` to specify which factor should be applied to the SCIP feasibility tolerance to initialize the primal feasibility tolerance of the LP solver.

- New parameter `lp/minmarkowitz` to set the Markowitz stability threshold. High values sacrifice performance for stability.

- Changed default of `propagating/redcost/useimplics` from TRUE to FALSE.

- See also the full release notes and the release paper.

- Symmetry detection and handling is now available on Windows, too.

- Updated Ipopt to 3.13, see above.

- Removed value `soplex2` for option `lp/solver`. `soplex` is now the default if no CPLEX license is available.

### 3.12.1.4.15 SHOT

- New solver by Andreas Lundell (Åbo Akademi University, Finland) and Jan Kronqvist (Imperial College London, UK).

- SHOT is a deterministic solver for mixed-integer nonlinear programming problems (MINLPs). It can solve convex and some nonconvex MINLP problems to global optimality. For other nonconvex problems, SHOT acts as a heuristic method, i.e., without providing guarantees of global optimality. For more information, see the GAMS/SHOT solver manual, the SHOT website, and the publications [126] [122].

- SHOT 1.0.0 (527f1a11) is available for Linux, Mac OS X (≥ 10.13), and Windows. It can be used with a GAMS base system license.
3.12.1.4.16  SoPlex

- New libraries 5.0.0 (6535a3c8).
  - Improvements in memory management.
  - The number of violations is now shown in the log.
- GAMS/SoPlex is now available for multithreaded solves.

3.12.1.4.17  XPRESS

- New libraries v8.8.3 (aka Optimizer 35.01.04). It has been a longer-than-usual time since the previous library update, so the list of updates and improvements enhancing performance, stability, and correctness is extensive and not reproduced here. For details, visit and search the FICO Community page.

3.12.1.5  Tools

3.12.1.5.1  GAMS IDE and GDXVIEWER

- As mentioned above, we switched the default application to be associated with GAMS files on Windows from the GAMS IDE to GAMS Studio with this release. The classic GAMS IDE has been shipped with the GAMS system for the last 20 years and is still the workhorse for many GAMS programmers. However, it does not provide all the features we see in modern development environments.
  Both the GAMS IDE and GDXVIEWER rely on a rather old software stack, which is hard to maintain with current operating systems and will become even harder over time. So we plan to drop the GAMS IDE and GDXVIEWER with an upcoming major release.

3.12.1.5.2  GAMS Studio

- New version 1.1.0.
  - Stability improvements.
  - Introduced new settings format.
  - Attention
    This required a reset of all settings. They are saved now in JSON format. As a consequence, Studio 1.0.0 cannot use settings that were written by Studio versions prior to 1.0.0 and vice versa.
  - Changed the default file encoding to UTF-8 for all platforms (this mostly affects Windows, there should be no change for other platforms).
  - Added option to change the default text encoding to Edit -> Encoding -> Select encodings.
  - Introduced Dark Theme for Windows and Linux as well (can be changed through Settings -> Editor).
  - Added editor for GAMS Configuration and menu entry for default GAMS Configuration file.
  - Added option to change display format of numerical values in GDX Viewer.
  - Changed location of default workspace and model library.
3.12.31 Distribution

- Changed location where gamslice.txt is put when created using the ”About GAMS” dialog to the new standard locations searched by GAMS (see installation notes for details).
- Improved shortcut layout:
  * Reintroduced ”Open Terminal” shortcut as Ctrl+T.
  * Introduced shortcut for ”Run MIRO base mode”: F8.
  * Introduced shortcut for ”Run MIRO hypercube mode”: Shift + F8.
  * Introduced shortcut for ”Run MIRO configuration mode”: Shift + F7.
  * Changed shortcut for focus ”Project Explorer” to Ctrl + 1.
  * Changed shortcut for focus ”Editor” to Ctrl + 2.
  * Changed shortcut for focus ”Parameter Editor” to Ctrl + 3.
  * Changed shortcut to extend and collapse ”Parameter Editor” to Ctrl + Alt + 3
  * Changed shortcut for focus ”Tab Browser” to Ctrl + 4.
  * Changed shortcut for focus ”Process Log” to Ctrl + 5.
  * Changed shortcut of duplicate line to Ctrl+D.
  * Changed shortcut of ”Toggle Comment” in ”Parameter Editor” to Ctrl + *
  * Changed shortcut of action ”Go to matching parentheses” to: Ctrl + B
  * Changed shortcut of action ”Select to matching parentheses” to: Ctrl + Shift + B
  * Fixed shortcut on Mac OS X: block edit mode is now CMD+Shift+Arrow Key.

3.12.1.5.3 GDXMERGE

- Fixed a problem when merging a GDX file with name .gdx.

3.12.1.5.4 GDXXRW

- Added new option dSetText to control the reading of explanatory text for set elements of domain sets.

3.12.1.6 Object Oriented APIs

- Option pySetup has been removed from the class GAMSOptions.
- Removed GAMS return code 116 (”Could not find GMSPython”) from the enumerate type representing the GAMS return code.
- Fixed a memory leak with GAMSModelInstance.solve.

3.12.1.6.1 C++

- As announced, dropped support for Microsoft Visual Studio 2013.

3.12.1.6.2 Python

- GAMS Python API is now distributed in source code under the MIT open source license.
- To access the GAMS API, the Python interpreter needs to find files in <GAMS Systemdirectory>/apifiles/Python/ and <GAMS Systemdirectory>/apifiles/Python/gams, where XY corresponds to the Python version X.Y, e.g. 3.7. The GMSPython installation is already set up to find all required files. The documentation section Let Python find the GAMS Python files gives details on how to use an external Python system with GAMS.
- There is a different directory structure in <GAMS Systemdirectory>/apifiles/Python. The content of the api directory has been moved to different places:
  - api_27 for the binary Python 2.7 API files,
  - src for the files required to build the expert-level API files from source,
  - gams for source files of the GAMS Python API plus a few extra Python source files for working with indexed EMP syntax, e.g. emppython1, and
  - thirdparty for the Python package ply for processing of indexed EMP syntax.
3.12.1.7 Expert Level APIs

- We plan to remove the GAMSX API with one of the next major releases.
- Added `palSetSystemName` to setup a generic auditline in PAL. Increased PAL API version to 3.

3.12.1.8 Model Libraries

3.12.1.8.1 GAMS API Library

- All Python examples have been reworked. Tests are performed also with the Python installation in `GMSPython`.
- New model `PBuildXPLevelAPI.gms`: Test building the expert-level APIs (57).

3.12.1.8.2 GAMS Data Library

New models:
- `rank01.gms`: Rank a vector, and display the data in sorted order (135)
- `rank02.gms`: Generate percentiles for a random vector (136)
- `rank03.gms`: Use `GDXRANK` to report multisectoral Monte Carlo results (137)
- `rank04.gms`: Repeated computation of percentiles within a loop (138)
- `rank05.gms`: Percentile ranking of household expenditure data with heterogenous household size (139)
- `GMSPythonCheck.gms`: Consistency check for `GMSPython` (140)

3.12.1.8.3 GAMS Model Library

- Updated model `encrypt.gms`: Input file encryption demo (318)
- New model: `waterld.gms`: Design of a Water Distribution Network with Limited Domain of Variables (426)

3.12.1.8.4 GAMS Test Library

New models:
- `gdxxrw15.gms`: GDXXRW - Testing option `dSetText` (810)
- `card03.gms`: Test card operator for scalar symbols (811)
- `limdom01.gms`: Test limited domains for variables in model (812)
- `limdom02.gms`: Test performance of limited domains for variables in model (813)
- `limdom03.gms`: Test the syntax of the model statement for limited domains for variables (814)
- `limdom04.gms`: Advanced test for limited domains for variables in model (815)
- `fnncpf3.gms`: Test correctness of `ncpf` intrinsic (816)
- `fnncpvupow2.gms`: Test correctness of `NCPVUpow` intrinsic (817)
- `fnncpvusin2.gms`: Test correctness of `NCPVUsin` intrinsic (818)
- `loop10.gms`: Test relaxed punctuation for control structures (819)
- `dirs01.gms`: Test search and processing of `gamslice.txt` and `gamsconfig.yaml` in user space (820)
### 3.12.1.8.5 PSOPT Model Library

- Updated model: *MultiperiodDCOPF24bus*: Multi-period DC-OPF for IEEE 24-bus network considering wind and load shedding

#### 3.12.1.9 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.13</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLNP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 7.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPLEX 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 35.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
3.12.2 31.1.1 Maintenance release (May 16, 2020)

3.12.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jan Abrell, Michael Ferris, Nick Sahinidis, and Karlo Šepetanc.

3.12.2.2 GAMS System

3.12.2.2.1 GAMS

- Fixed a bug where loading an empty universe from a GDX file caused an error.
- Fixed a bug in gamsinst that is used by the IDE to display available solvers.
- GAMS continues to search directories for the license and configuration file even if the construction of some directory names fails due to the absence of some environment variables, e.g. HOME.

3.12.2.2.2 GAMS Data Exchange (GDX)

- Fixed a bug where domain violations in higher dimensions might have been undetected when writing symbols to GDX.

3.12.2.3 Solvers

3.12.2.3.1 GUROBI

- Fixed a bug introduced with 31.1.0 where the GRB_LICENSE_FILE environment variable was incorrectly dropped.

3.12.2.3.2 MOSEK

- New libraries 9.2.7.

3.12.2.4 Tools

3.12.2.4.1 GAMS Studio

- New version 1.1.1.
  - Added command line parameter editor warnings for missing values.
  - Fixed problem with command line parameters not being updated when changing group.
  - Fixed behavior of Ctrl - End shortcut in process log and lst file view.
  - Fixed settings export not working when file already exists.
  - Fixed numpad not working when running Studio from an AppImage on Linux.
  - Fixed Shift+Arrow Key not working when running Studio from an AppImage on Linux.
3.12.2.5 Object Oriented APIs

- Fixed a bug in the Python API when exporting options (GAMSOptions.export()) that contain defines.

3.12.2.6 Model Libraries

3.12.2.6.1 GAMS Test Library

- Updated model dirs01.gms: Test search and processing of gamslice.txt and gamsconfig.yaml in user space (820)

3.12.3 31.2.0 Minor release (June 19, 2020)

3.12.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Salvador Doménech Martínez, and Kaushik Sinha.

3.12.3.2 GAMS System

3.12.3.2.1 GAMS

- The demo/community license limit of indexed external in- and output symbols for the GAMS/MIRO connector has been increased from 5 to 10. There is no limit on scalar external in- and output symbols. Starting with this version, one dimensional singleton sets are also not limited.

- Extend the influence of the command line parameter multiPass, so that it also ignores errors about unknown dimension with empty data statements when set to 2.

- Added three new put utility commands incMsg, incLog, and incMsgLog to include the content of a file into the log or listing file.

- Fixed a crash for CNS models which get solved asynchronously while asyncSolLst is set to 1.

3.12.3.3 Solvers

3.12.3.3.1 Gurobi

- Added option dumpBCSol to write incumbent solutions to GDX point files while Gurobi optimizes.

3.12.3.3.2 LocalSolver

- New libraries 9.5 (20200429).

3.12.3.3.3 Mosek

- New libraries 9.2.10.
3.12.3.3.4 SCIP

- New library 7.0 (01ae80d797).
- Fixed default setting for option misc/usesymmetry on Windows (was still set to 0, but should be 3).

3.12.3.3.5 SHOT

- New library 1.0.1.

3.12.3.4 Tools

3.12.3.4.1 GAMS Studio

- New version 1.1.2.
  - Performance improvements.
  - Stability improvements and minor bug fixes, e.g.,
    - Fixed crash when reopening a tab of a file that was removed from Project Explorer.
    - Fixed behavior of hotkeys to jump to beginning and end of line and page up/down.
    - Fixed Goto dialog jumping to beginning of file when entering invalid values.
    - Fixed shortcuts to focus Project Explorer and Editor not working when undocked.
    - Fixed that sizes of dock-widgets were not stored in FullScreen or Maximized window.

3.12.3.5 Object Oriented APIs

3.12.3.5.1 .NET

- Fixed a problem with reading labels which contain non-ASCII characters through GAMSSymbolRecord.Keys.

3.12.3.6 Expert Level APIs

3.12.3.6.1 C#

- Fixed a problem with reading arrays of strings which contain non-ASCII characters through different APIs.

3.13 30 Distribution

3.13.1 30.1.0 Major release (January 10, 2020)

3.13.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Todd Munson, Suguru Otani, Nick Sahinidis, and Dimitri Tomanos.
3.13.1.2 Platforms

- As announced, dropped support for Windows 7. The minimal required Windows version is now 8.1.
- Moved platform 32-bit MS Windows to peripheral platforms. We will completely drop support for 32-bit MS Windows with GAMS 31 (which was originally announced for GAMS 30).

3.13.1.3 Licensing and Demo Limits

- The GAMS demo limits have been increased. GAMS will generate linear (LP, RMIP, MIP) models up to 2000 constraints and 2000 variables and up to 1000 constraints and 1000 variables for all other model types. No other limits apply (e.g. non-zeros or discrete variables). Some solvers (e.g., the global solvers ANTIGONE, BARON, and LindoGlobal) might enforce additional restrictions.
- GAMS provides a free community license for users on request. This community license allows generating even larger models: up to 5000 constraint and variables for linear and 2500 constraints and variables for other model types. These free licenses are for demonstration and evaluation but not for commercial and production work, see the GAMS End User License Agreement.
- Starting with this release, GAMS requires a valid license even for the free demo system. Free demo licenses (time-limited for 12 months) can be generated at the download page. The free demo and community licenses require also a recent version (not older than 18 months) of GAMS to run. Please check the license documentation for details.
- GAMS Beta versions will stop working 90 days after their release. This is to prevent accidental use of beta software.

3.13.1.4 GAMS System

3.13.1.4.1 GAMS

- Added system suffix MACAddress, which contains the MAC address of the first network adapter.
- Allow to declare more than one parameter as table on a single line.
- Added new dollar control options $[on][off]Filtered: These control how GAMS loads data from GDX with $load, $loadR, and $loadM as well as from $offEmbeddedCode.
- Added new dollar control options $loadFiltered, $loadFilteredM, and $loadFilteredR to load data from GDX in filtered mode explicitly.
- Added new dollar control options $[on][off]ExternalInput to load data implicitly from an external source and $[on][off]ExternalOutput to save data implicitly to an external target.
- Added new command line parameters IDCGDXInput, IDCGDXOutput, IDCGenerateGDX, IDCGenerateGDXInput, IDCGenerateGDXOutput, IDCGenerateJSON, and IDCJSON.
- Embedded Python Code:
  - Changed the optional boolean argument domCheck to an enumerated option DomainCheckType in the gams.set method in embedded code. This might break existing code. Even if the domCheck argument was not used, the default behavior at execution time changes. In previous versions the content was domain checked, now it is read filtered.
  - The argument mergeType was documented incorrectly. With type MergeType.DEFAULT (or no argument given) GAMS merged the content, in contrast to what the documentation said (here replace was mentioned).
- Terminate with a proper error message if lines in the EMP info file exceed the maximum line length of 1020 characters.
- Fixed a problem where the order of a semicolon and certain dollar control options changed the behavior of the compiler.
3.13.1.4.2 Installer

- In addition to ..gms files, the Windows installer also associates .gdx files with either the GAMS IDE or GAMS Studio.

3.13.1.5 Solvers

3.13.1.5.1 BARON

  - Performance improvements for large-scale and mixed-integer quadratic problems.
  - Improvements in interface to IPOPT.

3.13.1.5.2 CBC

- New libraries.

3.13.1.5.3 CONOPT

- New library 4.16 with bug fixes.
  - Improvement in the transition from Phase 0 to Phase 1 and in the one-dimensional search close to optimality.

3.13.1.5.4 CPLEX

- New libraries 12.10.0.0.
- New options epLin and heuristicEffort.

3.13.1.5.5 GUROBI

- New libraries 9.0.0.
  - Gurobi 9 comes with a new bilinear solver, which allows to solve non-convex quadratic programming problems. The new parameter nonConvex needs to be set.
  - New general constraint types have been added to model functions like log, exp, sin, cos, and tan. New options funcPieces, funcPieceError, funcPieceRatio, funcPieceLength, and funcMaxVal have been added to control the piece-wise-linear discretization. The dot options doFuncPieces, doFuncPieceError, doFuncPieceRatio, and doFuncPieceLength allow control on an individual constraint basis.
- New parameters for new cutting planes: BQPCuts, relaxLiftCuts, and RLTcuts.
  - Fixed a problem when reporting the primal solution only.
  - Report the dual bound for continuous non-convex QP and QCP problems.
3.13.1.5.6 JAMS/LogMIP

- New option `userPFFile` to provide additional GAMS command line parameters for the GAMS run of the reformulated model.

3.13.1.5.7 LocalSolver

- New libraries 9.0 (20191219).

3.13.1.5.8 MOSEK

- New libraries 9.1.9.

3.13.1.5.9 ODHCPLEX

- New libraries 4.3.2 with bug fixes and performance improvements.

3.13.1.5.10 PATH

- New library 5.0.01 with bug fixes.
  - Fix seg fault when using the hidden option `output_hidden_options`.

3.13.1.5.11 SCIP

- New libraries 6.0 (90a56e0b).
  - Default for option `heuristics/rins/nodesquot` changed from 0.1 to 0.3.

3.13.1.6 Tools

3.13.1.6.1 GAMS Studio

- New version 0.14.1.
  - Stability improvements.
  - Added GDX Diff dialog to compare two GDX files. It can be opened by choosing `Tools > GDX Diff` from the menu or through the context menu off a GDX file in the Project Explorer.
  - Allow to open OS specific terminal in current working directory (`Tools > Terminal`) or in working directory of a specific group (using the context menu in the Project Explorer)
    * Note: This feature does not work on all Linux distributions, e.g., Ubuntu, yet.
  - Added basic integration of GAMS MIRO.
    * Note: This requires a separate installation of the GAMS MIRO package.
  - Improved GDX Viewer layout to optimize used screen space.
  - Adjusted layout of Reference File Viewer.
  - Added clear button to filter input field in GDX Viewer and Reference File Viewer.
3.13.1.7 Object Oriented APIs

3.13.1.7.1 C++

- We plan to drop support for Microsoft Visual Studio 2013 and maybe also 2015 with the next major release.

3.13.1.7.2 Python

- Added support for Python 3.8.
- As announced, dropped support for Python 3.4.

3.13.1.8 Expert Level APIs

- Added support for Python 3.8.
- As announced, dropped support for Python 3.4.

3.13.1.9 Model Libraries

3.13.1.9.1 GAMS Data Library  New models:

- pickstock.gms: Stock selection problem with MIRO (133)
- transport.gms: Classical transportation problem with MIRO (134)

3.13.1.9.2 GAMS Test Library  New models:

- embpy05.gms: Test merge/replace when loading data from embedded code (804)
- embpy06.gms: Test domain check/filtered when loading data from embedded code (805)
- load12.gms: Tests filtered load (806)
- idc01.gms: Implicit Data Contract Test (807)
- gurobi05.gms: GUROBI test suite - general constraints sin/cos (808)
- studio01.gms: Test Studio Startup (809)
### 3.13.1.10 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON 18.5.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.10</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.6</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.0</td>
<td>7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.1</td>
<td>11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 9.0</td>
<td>8.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.2 30.2.0 Minor release (February 07, 2020)

3.13.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Iswar Radhakrishnan, Simon Scolan, and Kirk Yost.

3.13.2.2 GAMS System

3.13.2.2.1 GAMS

- Fixed a problem when writing implicit data contract in JSON format (IDCGenerateJSON) with symbols that use a set more than once in its domain.
- Fixed a potential wrong domain violation error for symbols declared under $onExternalInput.
- Fixed gamsinst and the IDE to give better messages in case a license file is absent.
- GAMS incorrectly removed indexed semi-integer and semi-continuous variables that have been fixed at a level other than zero with holdFixed set to 1. This has been fixed.
- GAMS also removes scalar semi-integer and semi-continuous variables now, when they have been fixed at zero with holdFixed set to 1.
- Fixed an issue where protected symbols could be changed using certain option statements.
- Added new command line parameter IDCProtect to control assignment behavior of external input symbols at execution time: 0 allows assignment, 1 (default) protects external input symbols from being changed.
- Adjusted reading of GAMS command-line arguments on non-Windows platforms to allow for empty arguments and arguments with preceding and trailing blanks. Windows users will not be affected by this change, nor will users who do not use quotes to protect arguments. But these invocations will now work differently on Unix, and will give the same result as on Windows:
  - `gams main.gms --myPrefix "" user1 " four blanks here "`
  - `$call gams subjob.gms --cityA Tokyo --cityB ""`

3.13.2.3 Solvers

3.13.2.3.1 CONOPT

- New library 4.17 with bug fixes.
  - Fixed a problem where small infeasibilities were not correctly reported back to GAMS.
3.13.2.3.2 DICOPT

- Fixed a possible crash during exit when having used Ipopt as NLP solver.

3.13.2.3.3 GUROBI

- New libraries 9.0.1.
- Fixed a problem with tuning of lp/mps files. In contrast to the documentation, the tuning suite contained the model generated by GAMS when providing a list of models in lp/mps files to tune.
- Made value 3 available for Gurobi option `scaleFlag`, available since Gurobi 7.5.

3.13.2.3.4 MOSEK

- New libraries 9.1.11.

3.13.2.3.5 PATH

- New library 5.0.02.
  - Fix to two preprocessor issues: wrong solutions were being returned.

3.13.2.4 Tools

3.13.2.4.1 GAMS Studio

- New version 0.14.3.
  - Stability improvements.
  - New icon design.
  - Added filter facility for value columns in GDX Viewer.

3.13.3 30.3.0 Minor release (March 06, 2020)

3.13.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Andres J. Calderon, Antti Lehtila, Grégoire Pichenot, and Karlo Šepetanc.
3.13.3.2 GAMS System

3.13.3.2.1 GAMS

- Allow to create a secure work file on any platform for any target license platform.
- Fixed a bug which could have caused wrong domain violation errors with $\text{offEmbeddedCode}$.
- Symbols set in embedded Python code via the GAMS Python object oriented API (i.e. \texttt{gams.db[sym]}) were imported with \texttt{MergeType} ”replace” and \texttt{DomainCheckType} ”filtered”. This has been adjusted to both \texttt{MergeType} and \texttt{DomainCheckType} set to ”default”.
- Fixed a problem which was introduced with GAMS 30.1: The default for $\text{on}_\text{Filtered}$ should always be $\text{on}_\text{Filtered}$. This was not the case when restarting from an old work file.

3.13.3.3 Solvers

3.13.3.3.1 Antigone, Bonmin, Cbc, Couenne, Ipopt, SCIP

- The number of threads as specified by option \texttt{threads} was disregarded. This has been fixed.

3.13.3.3.2 CONOPT

- New library 4.18 with bug fixes.
- The degeneracy-breaking logic has been improved for numerically difficult models and the interaction between defined variables and initialized variables has been improved.

3.13.3.3.3 JAMS/LogMIP

- The maximum length of the directory name plus file name that GAMS can execute is 255. JAMS and LogMIP stopped working earlier. This has been fixed.

3.13.3.3.4 LocalSolver

- Added missing \texttt{lskeygen} tool for Linux system.

3.13.3.3.5 MOSEK


3.13.3.4 Tools

3.13.3.4.1 GAMS Studio

- New version 0.14.4.
  - Stability improvements.
  - Introduced Dark Mode support for macOS.
3.14 29 Distribution

3.14.1 29.1.0 Major release (November 15, 2019)

3.14.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, David Bernal Neira, Erwin Kalvelagen, and Scolan Simon.

3.14.1.2 Platforms

- As announced, dropped support for Mac OS X 10.12 (Sierra).
- After more than 20 years we plan to drop support for 32-bit MS Windows with the next major release. Some components, e.g., GAMS IDE and gdxxrw, will remain 32-bit executables in the 64-bit MS Windows GAMS distribution.
- We plan to drop support for MS Windows 7 with the next major release.

3.14.1.3 GAMS System

3.14.1.3.1 GAMS

- Extended the GAMS log to print out the Double Dash Parameters set in addition to the non-default Command Line Parameters.
- Added dollar control option $setDDList.Cont to allow to define a list of double dash parameters to be checked over multiple lines.
- Added new ways to execute external programs which implicitly check the errorLevel returned and raise a compilation or execution error if it is not zero, namely $call.checkErrorLevel, $hiddencall.checkErrorLevel, exec.execute.checkErrorLevel, put_utility exec.checkErrorLevel and put_utility shell.checkErrorLevel. This behavior can also be triggered without using the checkErrLevel suffix by setting $onCheckErrorLevel and checkErrorLevel, respectively.
- Added command line option captureModelInstance to capture all model instances generated and solved by a solve statement during a GAMS run.
- Allow to declare a parameter as table without entering data.
- Added new Dollar Control Options eval.Set, evalGlobal.Set, and evalLocal.Set to define a compile-time variable based on the content of a GAMS set.
- Removed limit of 10,000 columns in table declaration.
- Changed trace record fields of quadratic model types from historic to today's names (QP --> QCP, QMIP --> MIQCP, RQMIP --> RMIQCP).
- The gams.exe and gamskeep.exe programs on 64-bit Windows are now native 64-bit executables. Because of some technical requirements related to the Interrupt functionality, the IDE invokes GAMS via the new 32-bit program gams32.exe.
- Fixed wrong return string for %gams.action%.
- Fixed a problem with interval evaluation for Extrinsic Functions.
- Fixed a problem with GAMS parameter scriptExit when running from the object-oriented APIs, GAMS Studio, or GAMS IDE.
- Fixed a problem with trace record field ETSolve which reported NA always.
- Fixed a problem with potentially wrong explanatory text when implicit sets are loaded using $load.
3.14.1.3.2 Installer

- The DMG installer for Mac OS X has been reorganized and is now notarized, so that it should work on Mac OS X 10.15.
- The selfextracting archive for Mac OS X is now a 64-bit application.

3.14.1.4 Solvers

3.14.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2019.5 on Linux and Mac OS X.
- Updated Intel MKL libraries to version 2018.4 on Windows 64-bit again.

3.14.1.4.2 CBC

- New libraries.

3.14.1.4.3 CONOPT

- New library 4.15 with bug fixes.
- Option DF Method has been replaced by the options Tol_Def_Mult and Tol_Def_Init.

3.14.1.4.4 Convert

- Fixed a bug that caused a crash with option nlp2mcp selected.

3.14.1.4.5 DICOPT

- Fixed so called "integer cuts" when both binary and integer variables are present. DICOPT generated two integer cuts in this case, one that excluded the current values of binary variables and another one that excluded the current values of integer values. Now a single combined cut is generated, which allows that the values of the binary variables can repeat if the values of the integer values change and vice versa.
- Setting option weight to a value larger $10^{20}$ now disables the augmented penalty relaxation, that is, linearizations of nonlinear equations are added as hard constraints to the MIP relaxation instead of soft constraints.
- Enabling option convex now changes the default for option weight to maxdouble.

3.14.1.4.6 Examiner

- Added option objvarAutoAdjust to adjust objective variable to make objective equation feasible for models that allow objective reformulation.
3.14.1.4.7 Lindo/LindoGlobal

- New libraries 12.0.169.
- Added support for function `signPower`.

3.14.1.4.8 LocalSolver

- New libraries 9.0 (20191004) for Linux, Mac OS X, and 64-bit Windows.
  - Performance improvements on packing, routing, and scheduling problems as well as on nonlinear continuous problems.

3.14.1.4.9 LGO

- Fixed a bug in the computation of the best feasible solution - an uninitialized variable was used.

3.14.1.4.10 MOSEK

- New libraries 9.1.1.
- Correct rejection of problems with non-quadratic nonlinear objective function when constraints are linear.

3.14.1.4.11 SoPlex

- New libraries 4.0 (083219e0).

3.14.1.4.12 SCIP

- New libraries 6.0 (a15f6c41).

3.14.1.5 Tools

3.14.1.5.1 GAMS Studio

- New version 0.13.2
  - Stability and performance improvements.
  - Allow to adjust the numerical precision in the GDX Viewer.
  - Added dialog to create/update license file (pops up when a license is in the user's clipboard and "About GAMS" is opened).
  - Added full screen mode (shortcut on Windows and Linux: ALT+Return, shortcut on Mac OS X: META+CMD+F).
  - Added auto-resizing of columns when opening the Reference File Viewer.
  - Reworked process log to better handle long and fast output.
  - Improved behavior of F1 (Help) Key: Studio now opens widget specific help if available.
  - Renamed "Output View" to "Process Log".
- Added a tutorial about the usage of GAMS Studio to the documentation.
- As announced, dropped Studio for 32-bit Windows.
3.14.1.6 Object Oriented APIs

3.14.1.6.1 Python

- Added support for Python 3.7 on Windows 64-bit, Linux, and Mac OS X.
- We plan to drop support for Python 3.4 with the next major release.
- Support for Python 2.7 officially stops January 1, 2020. We plan to drop support for Python 2.7 with some future release. If you rely heavily on the availability and support of the GAMS API for Python 2.7 please contact support@gams.com to discuss your options.

3.14.1.6.2 Java

- Increase the minimum version requirement of the Java Runtime Environment to Java SE 8.
- The remove operation GAMSDatabaseIterator.remove is no longer supported. The method now throws an instance of java.lang.UnsupportedOperationException and performs no other actions. (To remove all records of the current iterating symbol, use the method GAMSSymbol.clear instead.)
- Fixed the behavior of the remove operation GAMSSymbolIterator.remove. The method now removes all records of the last GAMSSymbol element returned by an instance of GAMSSymbolIterator and can be called only once per call to GAMSSymbolIterator.next. The behavior of an iterator is unspecified if the underlying collection is modified while iterating.

3.14.1.7 Expert Level APIs

- Added support for Python 3.7 on Windows 64-bit, Linux, and Mac OS X.
- We plan to drop support for Python 3.4 with the next major release.
- We plan to drop support for Python 2.7 with some future release.

3.14.1.8 Model Libraries

3.14.1.8.1 GAMS Model Library

- Added inscribedsquare.gms: Inscribed Square Problem.

3.14.1.8.2 GAMS Test Library

- Extrinsic function test models: Upgraded query-library generation code ql.py to Python 3. The default for property "description" has been changed to be an empty string.

3.14.1.9 Solver/Platform availability matrix
<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>18.5.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.9</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.6</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 8.1</td>
<td>7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.1</td>
<td>11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 9.0</td>
<td>8.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soplex 4.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 33.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>32.01</td>
<td>29.01</td>
</tr>
</tbody>
</table>
3.15 28 Distribution

3.15.1 28.1.0 Major release (August 02, 2019)

3.15.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Johan Löfberg, Hans Mittelmann, and José Luis Vilar Zanón.

3.15.1.2 Platforms

- We plan to drop support for Mac OS X 10.12 (Sierra) with the next GAMS major release.

3.15.1.3 GAMS System

3.15.1.3.1 GAMS

- Added new dollar control option $\text{scratchFileName}$.
- Fixed a bug which caused a problem when unloading a multi-dimensional Alias at compile-time.

3.15.1.3.2 Documentation

- Improved ranking of results in offline search by minimizing the effect of the length of a document on the importance of a search term in that document.
- When searching the index, only relevant results are shown for subcategories as well.
- The full-text search now shows also results for a search in the index.

3.15.1.4 Solvers

3.15.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Downgraded Intel MKL libraries to version 2018.3 on Windows 64-bit.

3.15.1.4.2 ANTIGONE

- The timelimit is now applied to the wallclock time and wallclock time is reported in the log and back to GAMS.
3.15.1.4.3 BARON

- New libraries 19.7.13.
  - New relaxations and branching rules for nonconvex mixed-integer quadratic programs, leading to significant speedups for this class of problems.
  - Increased robustness in various range reduction routines.
  - Increased solvability of large-scale problems by BARON’s cut generators.
  - Enhanced convexity recognition routines.
  - Updated to IPOPT 3.12.13 and CBC 2.10.1.
- Fixed passing constant equations to BARON.

3.15.1.4.4 CBC

- Revised solver options.
  - Updated the documentation and possible values for many options, bringing them closer to the original Cbc option names and values.
  - The following options changed their type from boolean to string: crossover, cutoffConstraint, combineSolutions, Dins, DivingRandom, DivingCoefficient, DivingFractional, DivingGuided, DivingLineSearch, DivingPseudoCost, DivingVectorLength, feasPump, naiveHeuristics, pivotAndFix, randomizedRounding, Rens, Rins, roundingHeuristic, proximitySearch, presolve
  - For options dualPivot, primalPivot, and scaling value "auto" has been renamed to "automatic".
  - For option costStrategy, values "binaryfirst" and "binarylast" have been renamed to "01first" and "01last", respectively.
  - The following options have been added: factorization, denseThreshold, smallFactorization, sparseFactor, biasLU, psi, substitution, primalWeight, autoScale, bscale, gamma, KKT, expensiveStrong, OrbitalBranching, infeasibilityWeight, fixOnDj, sosPrioritize, cutLength, lagomoryCuts, latwomirCuts, hOptions, combine2Solutions, diveSolves, feasPump_artcost, feasPump_trachab, feasPump_cutoff, feasPump_increment, dwHeuristic, pivotAndComplement, VndVariableNeighborhoodSearch

3.15.1.4.5 CONOPT

- CONOPT3 accepts models with empty rows. The GAMS/CONOPT3 link sets the hidden option LSAZRW to true.

3.15.1.4.6 Convert

- Replaced the OSiL writer. The new writer supports more GAMS intrinsic functions and produces slightly smaller OSiL files with more compact expression trees.

3.15.1.4.7 IPOPT

- Fixed handling of Ipopt's feasible-point-found status.
3.15.1.4.8 LocalSolver

- Added option logfreq.

3.15.1.4.9 MOSEK

- New libraries 9.0.98.
  - Introduced the power and exponential cones, see also Conic Programming.
  - As announced, the optimizer for general convex nonlinear problems has been removed. Thus, solving a NLP, DNLP, MINLP, or RMINLP for which a conic reformulation is not recognized will now result in a missing capability solver status with GAMS/Mosek.
  - Improved presolve for particular conic problems.
  - Tighten the stopping criteria when solving conic optimization problems: Changed default of MSK_DPAR_INTPNT_CO_TOL_INFEAS from 1e-10 to 1e-12, of MSK_DPAR_INTPNT_CO_TOL_REL_GAP from 1e-7 to 1e-8, and of MSK_DPAR_INTPNT_QO_TOL_INFEAS from 1e-10 to 1e-12. However, note that one can let Mosek relax feasibility and optimality tolerances if it cannot make sufficient progress, see MSK_DPAR_INTPNT_CO_TOL_NEAR_REL and MSK_DPAR_INTPNT_QO_TOL_NEAR_REL.
  - Changed scaling of interior-point optimizer so better accuracy is obtained in some cases.
  - Introduced an outer approximation method for solving conic mixed integer optimization problems. This can be enabled with the parameter MSK_IPAR_MIO_CONIC OUTER APPROXIMATION.
  - Much improved performance on recent AMD CPUs. For linear algebra, the BLIS library is employed when run on AMD CPUs.
  - Better performance on CPUs that support AVX-512 instructions.
  - Removed parameters: MSK_DPAR_DATA_TOL_AIJ, MSK_DPAR_INTPNT_NL_*
  - Further new parameters: MSK_IPAR_PRESOLVE_MAX_NUM_PASS, MSK_IPAR_PRESOLVE_MAX_NUM_PREP, MSK_IPAR_SIM_SEED, MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS, MSK_IPAR_MIO_MAX_NUM_ROOT, MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT, MSK_IPAR_MIO_SEED, MSK_IPAR_MIO_FEASPUMP_LEVEL, MSK_IPAR_WRITE_COMPRESSION, MSK_IPAR_OPF_WRITE_LP, MSK_IPAR_PTF_WRITE_TRANSFORM
  - Changed default of MSK_DPAR_MIO_TOL_ABS_GAP from 0 to value of GAMS option OptCA (was used for MSK_DPAR_MIO_NEAR_TOL_ABS_GAP before) and of MSK_DPAR_MIO_TOL_REL_GAP from 0 to value of GAMS option OptCR (was used for MSK_DPAR_MIO_NEAR_TOL_REL_GAP before).

- Fixed handling of Mosek certificates for primal and dual infeasibility, see Infeasible/Unbounded Models.

3.15.1.4.10 SCIP

- New libraries 6.0 (f79421d).
3.15.1.5 Tools

3.15.1.5.1 GAMS Studio

- New version 0.12.3.
  - Stability and performance improvements.
  - Added GAMS Solver Option Editor to view and edit a solver-specific option file.
  - Renamed Option Editor to Parameter Editor to better distinguish solver options and GAMS parameters.
  - Added menu entry GAMS -> Delete scratch directories to delete all scratch directories in current working directory. A similar dialog will also be triggered if GAMS returns 110 (“Too many scratch directories”).
  - Added file associations and icons for .gms and .gdx files on Mac OS X.
  - Changed shortcut to focus Main widget from Ctrl+H to CTRL+E (there was a collision on Mac OS X).
  - Improved keyboard navigation between files and widgets.
- We plan to drop Studio for 32 bit Windows with the next major release.

3.15.1.6 APIs

- As announced, dropped support for Python 2.6.
- We plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.

3.15.1.7 Model Libraries

3.15.1.7.1 GAMS Model Library

- Changed model sigma in immun to minimize standard-deviation instead of variance.
- Added missing nonnegativity requirement on variables in model wall.

3.15.1.8 Solver/Platform availability matrix

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECPC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>18.5.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>x86 32bit MS Windows</td>
<td>x86 64bit MS Windows</td>
<td>x86 64bit Linux</td>
<td>x86 64bit Mac OS X</td>
<td>Sparc 64bit SOLARIS</td>
<td>IBM Power 64bit AIX</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>-----------------</td>
<td>--------------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.9</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Gurobi 8.1</td>
<td>7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.0</td>
<td>11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 8.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SolveEngine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 4.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 33.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3.15.2 28.2.0 Minor release (August 19, 2019)

3.15.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Nayee Chowdhury, Salvador Doménech Martínez, Michael Ferris, Antti Lehtila, Evangelos Panos, Jochen Uhrich, and Alexey Ziuzin.
3.15.2.2 GAMS System

3.15.2.2.1 GAMS

- Setting $\text{onMultiR}$ allows the redefinition of macros as well now.
- Fixed a problem finding the Python interpreter, that comes with GAMS, in certain cases.
- Fixed a bug in the model generation, which could have caused a crash or wrong results in some cases.

3.15.2.3 Solvers

3.15.2.3.1 CPLEXD

- Corrected reporting of solution for (MI)QCP problems when using feasopt.
- CplexD now reports the objective of the feasible relaxation in the solution summary when using feasopt.

3.15.2.4 Tools

3.15.2.4.1 GAMS Studio

- New version 0.12.4
  - Stability and performance improvements.
  - Improved robustness of glb file parser (see User Libraries).
  - Added "What's new" menu entry to show latest changes of Studio.

3.15.2.5 Object Oriented APIs

- Fixed a problem, where GAMSDatabase.Export triggered an exception because of domain violations, even if only relaxed domains were used.

3.15.2.5.1 Java

- Added new method GAMSSymbolRecord.dispose for on-demand release of external resources hold by non-java library.

3.16 27 Distribution

3.16.1 27.1.0 Major release (April 24, 2019)

3.16.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Arne Drud, Jan-Erik Justkowiak, Renger van Nieuwkoop, and Manuel Wetzel.
3.16.1.2 Platforms

- As announced, support for Mac OS X 10.11 (El Capitan) has been dropped.
- As announced, support for Windows Vista has been dropped.

3.16.1.3 GAMS System

3.16.1.3.1 GAMS

- **Embedded Python Code:**
  - Allow Implicit Set Definition (or: Domain Defining Symbol Declarations) now for symbols loaded from embedded code as well.
  - Changed the default behavior of $offEmbeddedCode and endEmbeddedCode: Previously, internal resources got freed and with the next start of an embedded Python code section, the Python environment had to be reinitialized. This is not the case anymore. Also, the interpreter stays "alive" now, which allows to access Python symbols defined in one embedded code block in a following block. The old behavior could still be activated by setting the new command line parameter freeEmbeddedPython to 1.
  - Changed the optional boolean argument merge to an enumerated option mergeType in the gams.set method in embedded code. This might break existing code. Even if the merge argument was not used, the default behavior at compile time changes. In previous versions the content of an already defined symbol was quietly replaced while GAMS now triggers a compilation error 194 Symbol redefined.
  - Fixed a bug in embedded code when setting pySetup to 0. Additional steps are required on Linux and Mac OS X to use an alternative Python installation. Details can be found in section Porting to a Different Version of Python.
  - A GAMSDatabase created in embedded code via gams.ws.add_database(source_database = gams.db) had all the symbols but no data. This has been fixed.
  - gams.db.export(file_name) in embedded code resulted in an error for scalar symbols without data. This has been fixed.
  - Fixed a problem with embeddedCode/continueEmbeddedCode after an assignment that was not finished with a ; like in this example:
    ```plaintext
    Set i /1*3/;
    Parameter xl(i);
    xl(i) = uniform(0.5,1)
    embeddedCode Python:
    print(list(gams.get('xl')))
    endEmbeddedCode
    Display xl;
    ```
    This leads to an error now. In the past, the assignment in line 3 was executed after the embeddedCode block leading to unexpected results.

- Extended the GAMS log to print out the non-default Command Line Parameters.

- Improved performance of execute/unload when option gdxUELs is set to full.

- Improve performance of model generation when multi-dimensional variables are used, where the controlling set for dimension n>1 is big, e.g.:

  ```plaintext
  Set tiny /1*2/, huge /1*100000/ ;
  Variable x(i,j); Equation e;
  ...
  e.. sum((tiny,huge), x(tiny,huge)) =e= 42;
  ```
• Fixed a potential crash when more than 250 million labels are used in one model.
• Fixed a bug with Implicit Set Definition: When used with $onMulti to merge data into a non-empty set, it could have happened that an element was added twice in some cases.
• Fixed a bug causing a broken dump file if $onVerbatim and $ifThen was used with DumpOpt.

3.16.1.3.2 Documentation
• Added documentation for the put utility keyword assignText (which has been in the system since 24.6).

3.16.1.3.3 GDX library
• Improve the behavior of GDX when handling large data sets (e.g. writing a set with 210 million elements, each having its own lengthy set text). Performance is better - less memory and less time are required. The limits on the amount of data that can be handled have been extended somewhat, and when these limits are reached this is handled more gracefully than previously - with a helpful message instead of a segmentation fault or erroneous results.
• Introduce consistent handling of the empty string vis-a-vis set text for UELs. This is only relevant for user applications that use the low-level GDX API in an unsupported way.
  − Previously the proper usage was to avoid adding the empty string to the list of set text strings. Applications that broke this rule would produce a GDX file with an abnormal internal structure. This structure may lead to jumbled set text.
  − With the updated GDX, it is not necessary to avoid adding an empty string to the set text list.
  − Applications holding to previous proper usage (i.e. those that avoid adding the empty string to the list of set text strings) will produce good GDX files regardless of GAMS version, and these files will behave as expected regardless of the GAMS version in which they are used.

3.16.1.4 Solvers
3.16.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP
• Updated Intel MKL libraries to version 2019.3 on Linux and Mac OS X and to 2018.4 on Windows 64-bit.

3.16.1.4.2 BARON
• New libraries 19.3.24.
  − Improved automatic differentiator that is up to two orders of magnitude faster for large-scale problems.
  − Improved data structures and cut pool manager to reduce memory footprint and time requirements for large-scale problems.
  − Improvements in various components, including parallel threads and BARON's IPOPT interface for Mac OS X.
  − New tree management algorithms.
  − Improved range reduction and probing algorithms.
  − Using Intel MKL for faster BLAS and LAPACK routines.
  − Ipopt now uses METIS for linear system partitioning.
  − New option FirstLoc to terminate BARON as soon as a local optimum is found.
• New option ClockType to determine whether wall-clock or CPU time should be reported back to GAMS.
3.16.1.4.3 BDMLP and SBB

- The definition of SOS1 and SOS2 variables has changed. BDMLP and SBB used to count the number of elements in a set that are off the lower bound. BDMLP and SBB were the only two solvers in the GAMS solver portfolio with this definition. All other solvers counted elements that are off zero. The definition of SOS1 and SOS2 variables in BDMLP and SBB has been changed to the off zero definition to make their behavior more consistent with other GAMS solvers.

3.16.1.4.4 CBC

- New libraries 2.10.
  - Improved handling of SOS, starting point, and symmetries.
  - Improved performance of primal heuristics regarding the handling of implicit integer variables.
  - Mini-B&B is now disabled when solving with multiple threads.
  - Changed default value for zero half cuts parameter from off to ifmove.

3.16.1.4.5 CONOPT 4

- New library 4.11 with improvements in reliability and efficiency for large and difficult models.
- New option Tol_Opt_LinF: Optimality tolerance when infeasible in Linear Feasibility Model.

3.16.1.4.6 CPLEX/CPLEXD

- New libraries 12.9.0.
  - New options: CPLEXfolding, CPLEXwriteprob (CplexD only), CPLEXwarninglimit.
  - New hierarchical multi-objective optimization (CplexD only): CPLEXmultobj, CPLEXmultobjmethod, CPLEXobjnabstol, CPLEXobjnreltol.
  - The symmetry breaking parameter CPLEXsymmetry now applies only to MIP models. Use the new CPLEXfolding parameter for LP models.

3.16.1.4.7 GUROBI

- New libraries 8.1.1.

3.16.1.4.8 JAMS

- Experimental indexed EMP syntax has been updated: It can now take all the equilibrium related keywords (max, min, vi, qvi, dualvar, visol, and implicit) and shared constraints and variables.

3.16.1.4.9 LocalSolver

- New libraries 8.5 (20190130).
  - Strong lower bounds based on nonlinear relaxation techniques coupled with innovative branch-and-bound heuristics.
3.16.1.4.10 Mosek

- New libraries 8.1.0.80.

3.16.1.4.11 ODHCPLEX

- New libraries 4.23.

3.16.1.4.12 OQNLP

- As announced, OQNLP has been dropped.

3.16.1.5 Tools

3.16.1.5.1 CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.
- New library 6.2.0 on Mac OS X.

3.16.1.5.2 GAMS Studio

- New version 0.11.1.
  - Stability and performance improvements.
  - Introduced a toolbar to replace the "Execution and Option Parameter Editor".
  - Editor
    * Improved the syntax highlighting.
    * Allow to open huge read-only (lst) files without significant memory requirement.
    * Added bookmark support.
    * Added Ctrl+Up/Ctrl+Down as shortcut for line wise scrolling without moving the text cursor.
    * Added Ctrl+Home/Ctrl+End as shortcut for page wise scrolling without moving the text cursor.
    * Added warning pop-up when trying to open a >50 Mb file in edit mode.
  - Various
    * Added Ctrl+Alt+L to extend/collapse the extended option editor
  - Forced light application theme on Mac OS X to circumvent problems using the Dark Mode of Mac OS X Mojave.

3.16.1.5.3 GDXDUMP

- New command line option setText to show the list of set text stored in a GDX file.
- Option symbols now lists the cardinality of a symbol in the column Records.
3.16.1.5.4 GDXXRW

- Fixed exit behavior if there were some duplicate records found when reading a symbol but less than specified by `maxDupeErrors`. If there are no other errors the exit code is 0 now (it was 16 in the past).

3.16.1.5.5 GMSZIP/GMSUNZIP

- Replaced the 32bit binaries `gmszip` and `gmsunzip` by more recent 64bit versions on Mac OS X.

3.16.1.5.6 MPS2GMS

- Turn some errors that can be safely ignored when reading an MPS file into warnings.

3.16.1.6 Object Oriented APIs

3.16.1.6.1 Python

- Fixed a bug with default variable levels being 1.0 instead of 0.0.

3.16.1.7 Model Libraries

3.16.1.7.1 GAMS Data Library

- `GDX2ACCESSExample1.gms` : Dumping the Contents of `transport.gdx` (125)
- `GDX2ACCESSExample2.gms` : Writing Explanatory Text to Database (126)
- `GDX2ACCESSExample3.gms` : Dumping a large Table to Database (127)
- `GDX2ACCESSExample4.gms` : Special Value Mapping (128)
- `GDX2ACCESSExample5.gms` : Renaming Fields (129)

3.16.1.7.2 GAMS Test Library

- `sosmiqcp01.gms` : SOS1 and SOS2 behavior - MIQCP (794)
- `sosmip01.gms` : SOS1 and SOS2 behavior - MIP (795)
- `sosminlp01.gms` : SOS1 and SOS2 behavior - MINLP (796)

3.16.1.8 Solver/Platform availability
<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>18.5.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.9</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.6</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 8.1</td>
<td>7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.1</td>
<td>11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 8.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPLEX 4.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 33.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>32.01</td>
<td>29.01</td>
</tr>
</tbody>
</table>
3.16.2 27.2.0 Minor release (May 23, 2019)

3.16.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Michael Burkhhardt, and Karlo Šepetanc.

3.16.2.2 GAMS System

3.16.2.2.1 GAMS

- Fix problem with missing entries for scalar variables and equations in the index file for the GAMS Output in certain cases.

3.16.2.3 Solvers

3.16.2.3.1 CONOPT

- New library 3.17J.
- New library 4.12.

3.16.2.3.2 Lindo/LindoGlobal

- New libraries 12.0.157.

3.16.2.3.3 LOCALSOLVER

- New libraries 8.5 (20190430).

3.16.2.3.4 NLPEC

- Added new option parmFile to allow to point to additional options for the GAMS run of the scalar model produced by NLPEC.

3.16.2.3.5 ODHCPLEX


3.16.2.3.6 OSIXPRESS

- Fixed use of standalone Xpress license.
3.16 27 Distribution

3.16.2.4 Tools

3.16.2.4.1 GAMS Studio

- New version 0.11.2.
  - Stability and performance improvements.
  - GDX Viewer
    * Added Drag and Drop feature in GDX Table View

3.16.2.5 Object Oriented APIs

3.16.2.5.1 Python

- Fixed a memory leak in the Python 3 version of the API.
- We plan to drop support for Python 2.6 with the next major release.

3.16.2.6 Expert Level APIs

- Fixed a memory leak in the Python 3 version of the APIs.
- We plan to drop support for Python 2.6 with the next major release.

3.16.3 27.3.0 Minor release (July 04, 2019)

3.16.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Erwin Kalvelagen, Wolfgang Kuehn, Bruce McCarl, and Michael Winkler.

3.16.3.2 GAMS System

- Fixed a bug that resulted in undercounting the number of physical and logical processors on Windows systems with more than 64 logical processors. GAMS/Base and many solvers use this count, e.g. when setting threads to 0 or -1.

3.16.3.2.1 GAMS

- Allow domain projection with $load also if the source symbol is a variable or equation.
- Fixed the cause for an unwanted compilation error with embeddedCode in certain situations.
- Unified behavior of the profile option: In the past, the Profile Summary was always written, when the profiling was activated from the command line, but was sometimes omitted when set as an option. Now, the summary gets always written. It does not matter anymore, where profiling was activated.
3.16.3.2.2 Documentation

- Added a Preface that reflects the history of the GAMS documentation. In particular it points out
  the importance of external contributors, in particular Bruce McCarl.

3.16.3.3 Solvers

3.16.3.3.1 BARON

- Fixed load of Xpress library on Linux.

3.16.3.3.2 CBC

- New libraries.

3.16.3.3.3 CONOPT

- New library 3.17K.
- Bug fix in library 4.12.

3.16.3.3.4 GAMSCHK

- Fixed error in postopt report. Previously, the report for the column occurring last in the listing file
  would be omitted, and all its entries would appear in the report for the preceding column.
- Fixed crash that occurred in some cases.
- The above problems occurred in GAMS 26 and previous. A fix was introduced with GAMS 27, but
  this fix went too far and introduced new problems. With this version, the fix to 27.1 is reverted and
  a more moderate fix is used instead.

3.16.3.3.5 LocalSolver

- Fixed missing solver log.

3.16.3.3.6 Mosek

- New libraries 8.1.0.81.
- Fixed ignoring option solvefinal.

3.16.3.3.7 SCIP

- New libraries 6.0.2.
3.16.3.3.8 SoPlex

- New libraries 4.0.2.

3.16.3.4 Tools

3.16.3.4.1 GAMS Studio

- New version 0.11.5
  - Stability and performance improvements.
  - Added buttons to toolbar to show/hide Project Explorer, Output-, and Help widget.
  - Added shortcut Ctrl+H to focus Main widget.
  - Added shortcut Ctrl+Shift+G to focus Output widget.
  - Added shortcut Esc to close several widgets.
  - GDX Viewer:
    * Columns in the data view are now resized to the minimum width necessary to see all the data automatically.
    * Added shortcut to auto-resize all columns in the data view after the width was changed by hand (Ctrl+R).
  - Search:
    * Added shortcut Ctrl+Enter for "Search All".
    * Added ability to replace in more than one file at a time.
    * Find Next/Previous now respects all search options and will jump to the next file if there are also matches.

3.16.3.5 Object Oriented APIs

3.16.3.5.1 Python

- Fixed a bug in GamsJob.interrupt() that caused an OverflowError on 64 bit platforms.

3.17 26 Distribution

3.17.1 26.1.0 Major release (February 02, 2019)

3.17.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Stefano Alva, Adam Christensen, Hanna Donau, Stephen Frank, Anastasis Giannousakis, Jan-Erik Justkowiak, David Laudy, Andreas Lundell, Thomas Maindl, Nils Mattus, Scott McDonald, Noah Rhodes, Tom Rutherford, and Anna Straubinger.
3.17.1.2 Versioning

From this release on, we use a new interpretation on the GAMS version number scheme XX.Y.Z. XX alone now indicates the GAMS distribution number (was XX.Y before). Y is used to distinguish between beta, major, and minor releases of the XX distribution. Z is used to distinguish different maintenance releases. That is, we now use

- XX.0.Z for beta versions,
- XX.1.0 for the major release of distribution XX,
- XX.Y.0, Y > 1, for minor releases,
- XX.Y.Z, Y ≥ 1, Z > 0, for maintenance releases.

See Release Types regarding the meaning or what constitutes a major, minor, or maintenance release. As before, we increase the required license date for major releases only.

Further, we changed the default installation directory of the GAMS system from using only the distribution number to also include the minor version number. Thus, as before, GAMS is now installed into directory names which contain the XX.Y part of the GAMS version number.

3.17.1.3 Platforms

- We plan to drop support for Mac OS X 10.11 (El Capitan) with the next major release.
- We plan to drop support for Windows Vista with the next major release.
- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

3.17.1.4 GAMS System

3.17.1.4.1 GAMS

- Changed behavior of $compress, $decompress, and $encrypt: source and target cannot be identical anymore; this could have lead to unreliable behavior in the past.
- Extended the influence of $onMulti: This affects the model statement as well now.
- Added new dollar control option $onMultiR, which is similar to $onMulti but replaces existing data instead of merging into it.
- Added put_utility keyword solver to select a solver for a given or all possible model types by name at execution time.
- Extended the syntax to allow Implicit Set Definition (or: Domain Defining Symbol Declarations). Here is a small example showing how this can be used:

```plaintext
Set
   i 'canning plants'
   j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
          new-york chicago topeka
seattle    2.5    1.7    1.8
san-diego  2.5    1.8    1.4;

Display i,j;
```
And this is the result of the `Display` statement:

```plaintext
---- 10 SET i canning plants
    seattle , san-diego

---- 10 SET j markets
    new-york , chicago , topeka
```

- Updated the base module and components as necessary to allow for spaces in names of any files and directory used by GAMS, including the system directory, the working directory, the scratch directory, the process directory, and remove any checks warning about and preventing this. While the Windows system had already allowed spaces in previous versions, this change updates the other platforms to behave similarly.

- Changed the reading of command-line arguments on non-Windows platforms! This should only affect how arguments with spaces in them are passed on to GAMS. Previously, such arguments needed extra quoting or protection on non-Windows platforms. With this update, the extra protection is no longer needed: the same quoting of space-containing arguments can be used on all platforms. For example, with GAMS 25.1 and previous, two pairs of quotes were required on non-Windows platforms to protect spaces in an argument:

  ```plaintext
gams xx.gms --MY_GDX '"outer space"'
```

while now only one pair of quotes is needed on all platforms:

```plaintext
gams xx.gms --MY_GDX "outer space"
```

- Increased the number of labels, which can be handled by GAMS from 200 million to \(\sim 2.1\) billion and let GAMS terminate with the meaningful error message, if that limit is reached.

- Windows only: Fixed a bug which treated a path (e.g. to an input file or used as `inputDir`) as relative path, if it starts with `\/`. Example: When running `cd C:\tmp & gams.exe /data/myModel.gms`, GAMS tried to load the file `C:\tmp\data\myModel.gms` in the past. Now, it loads `C:\data\myModel.gms`.

- Fixed some problems with option `asyncSolLst`.

- Fixed a bug which could have caused GAMS to accept duplicate entries in a data statement, when they came in unsorted order.

- Fixed the dollar control option `$onUpper` (it did not do anything before).

### 3.17.1.4.2 GMSPython

- Updated `pip` to work with recent Mac OS X versions.

### 3.17.1.5 Solvers

#### 3.17.1.5.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Fixed an MKL error when running any of these solvers after CPLEX with `solvelink=5` in the same GAMS run on Linux.

- Updated Intel MKL libraries to version 2019.0 on Linux and to version 2018.3 on Windows 64-bit.
3.17.1.5.2 BARON

- New libraries 18.11.12 for 64-bit platforms.
  - New cutting planes for convex-transformable functions.
  - New cutting planes for polynomial optimization problems.
  - More robust treatment of cutting planes.
  - Improved optimality-based range reduction mechanisms.
  - CBC can now use multiple threads on Windows, too.
- We plan to drop BARON for Windows 32-bit with some future release.
- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time.

3.17.1.5.3 Bonmin, CBC, Couenne, Ipopt

- New libraries.

3.17.1.5.4 CONOPT 4

- New library 4.09 with many reliability improvements and a fix for wrong non-opt markers.

3.17.1.5.5 CPLEX

- Ctrl-C (or Interrupt) was broken for GAMS/CplexD. This has been fixed.
- Improved performance of retrieving the solution for a QCP with many quadratic rows.

3.17.1.5.6 Gurobi

- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time on Unix.
- Allow quadratic rows with =E= and let Gurobi deal with potential errors. Gurobi can sometimes substitute quadratic terms defined by =E= rows. The check of quadratic row to be either =L= or =G= in the GAMS/Gurobi link prevented this.

3.17.1.5.7 KESTREL

- Added support for MPSGE models.

3.17.1.5.8 Knitro

- New libraries 11.1.1 (11.0.1 for 32-bit Windows) containing several bug fixes and performance improvements, including:
  - improved performance on convex quadratic models,
  - improved generation of knapsack cuts, resulting in faster performance on some MINLP models,
  - improved performance when using the SQP algorithm,
  - new values for option linSolver to choose parallel factorization routines MA97 and MA86.
- Artelys has announced that Knitro 11.0 was the last Knitro release to support 32-bit Windows. We will be dropping GAMS/Knitro for 32-bit Windows in the near future.
3.17.1.5.9  **Lindo/LindoGlobal**

- New libraries 12.0.90.
  - LP Solver Improvements:
    - With new enhancements made to the simplex solvers, the average performance on large instances has increased by 18% for the primal simplex and 15% for the dual simplex compared to the previous version.
    - Improved performance on LP’s when using multiple cores with concurrent execution of Primal, Dual, and Barrier.
  - Quadratic and Nonlinear Solver Improvements:
    - Much faster handling of large quadratic matrices, e.g., 1000 x 1000.
  - Global Solver Improvements:
    - Improved handling of discontinuous functions, e.g., MOD(x,k), ROUND(x).

3.17.1.5.10  **LocalSolver**

- New libraries 8.0 (20181106).
  - Performance improvements on routing & scheduling problems, especially Pickup & Delivery problems.
  - Learning algorithms inside LocalSolver, which allow to automatically and dynamically tune up the search process for each solved instance, were improved. This leads to improved performances, especially for short resolution times (minutes).

3.17.1.5.11  **Mosek**

- New libraries 8.1.0.72.

3.17.1.5.12  **ODHCPLEX**

- New libraries 4.09 for Linux with many reliability improvements.
- New libraries 4.13 for Windows 64-bit with many reliability improvements and addressing dynamic loading in concurrent threads and processes.
- Updated options: *DynamicSearch*, *FirstFeas*, and *Strategy*

3.17.1.5.13  **OQNLP**

- We plan to drop OQNLP with the next major release.
3.17.1.5.14 PATH

- New libraries 5.0.00.
  - Fixed crash in Lemke's method that could happen for models with extremely poor scaling.
  - New capability to use alternate basis-handling packages via dynamic loading of shared libraries. Two alternate packages are currently supported:
    - BLU-LUSOL: Block LU updating using the LUSOL routines. The shared library required for this is included in the distribution.
    - UMFPACK: Tim Davis’ multifrontal LU factorization package. The shared library required for this is not included in the GAMS distribution, but can be downloaded from the SuiteSparse site. N.B.: We have verified UMFPACK on Linux, Mac OS X, and Windows using UMFPACK v5.7.7 from SuiteSparse 5.3.0.

3.17.1.5.15 SCIP

- New libraries 6.0.1.
  - Primal Heuristics
    * new diving heuristic farkasdiving that dives into the direction of the pseudosolution and tries to construct Farkas-proofs
    * new diving heuristic conflictdiving that considers locks from conflict constraints
    * performance improvements for Adaptive Large Neighborhood Search
      - changed default of parameter heuristics/alns/adjustminimprove from 1 to 0
      - changed default of parameter heuristics/alns/alpha from 0.2 to 0.0016
      - new parameter heuristics/alns/adjusttargetnodes
      - changed default of parameter heuristics/alns/eps from 0.5 to 0.468584
      - changed default of parameter heuristics/alns/gamma from 0.2 to 0.0704146
      - changed default of parameter heuristics/alns/minimprovehigh from 0.1 to 0.01
      - changed default of parameter heuristics/alns/minimprovevelow from 0.0001 to 0.01
      - removed parameter heuristics/alns/stallnodefactor
      - changed default of parameter heuristics/alns/startminimprove from 0.05 to 0.01
      - changed default of parameter heuristics/alns/targetnodefactor from 1.5 to 1.05
      - new parameter heuristics/alns/unfixtol
      - changed default of parameter heuristics/alns/crossover/minfixingrate from 0.4 to 0.3
      - changed default of parameter heuristics/alns/dins/maxfixingrate from 0.5 to 0.9
      - changed default of parameter heuristics/alns/dins/minfixingrate from 0.1 to 0.3
      - changed default of parameter heuristics/alns/localbranching/minfixingrate from 0 to 0.3
      - changed default of parameter heuristics/alns/mutation/minfixingrate from 0.4 to 0.3
      - changed default of parameter heuristics/alns/proximity/minfixingrate from 0 to 0.3
      - changed default of parameter heuristics/alns/rens/maxfixingrate from 0.7 to 0.9
      - changed default of parameter heuristics/alns/rins/maxfixingrate from 0.6 to 0.9
      - changed default of parameter heuristics/alns/rins/minfixingrate from 0.2 to 0.3
      - changed default of parameter heuristics/alns/zeroobjective/minfixingrate from 0 to 0.3
  - New branching rule lookahead that evaluates potential child and grandchild nodes to determine a branching decision
  - LP Solver Interface
    * lp/checkstability is now properly implemented for use with SoPlex
    * new parameter lp/alwaysgetduals
3.17.26 Distribution

- new parameter lp/checkfarkas
  - Separation (cutting planes)
    - cuts generated from certain quadratic constraints with convex feasible region are now global
    - new parameter separating/dircutoffdistfac
    - new parameter separating/efficacyfac
    - removed parameter separating/maxincrounds
    - new parameter separating/zerohalf/dircutoffdistweight
    - new parameter separating/zerohalf/efficacyweight
    - changed default of parameter separating/zerohalf/goodscore from 0.9 to 1
    - new parameter separating/zerohalf/goodmaxparall
    - new parameter separating/zerohalf/initseed
    - new parameter separating/zerohalf/maxparall
    - new parameter separating/zerohalf/objparalweight
  - Symmetry Handling
    - restructured timing of symmetry computation to allow to add symmetry handling components within presolving
    - changed default of parameter constraints/symresack/ppsymresack from 0 to 1
    - new parameter presolving/symbreak/addconsstming
    - changed default of parameter presolving/symbreak/detectorbitopes from 1 to 0
    - removed parameter "presolving/symmetry/computepresolved"
    - new parameter propagating/orbitalfixing/enabledafterrestarts
    - new parameter propagating/orbitalfixing/performpresolving
    - changed default of parameter propagating/orbitalfixing/presolpriority from 0 to -1000000
    - changed default of parameter propagating/orbitalfixing/presoltimeing from 28 to 16
    - new parameter propagating/orbitalfixing/symcomptiming
    - improved bound tightening for some quadratic equations
    - new parameter display/relevantstats
    - new parameter misc/scaleobj
    - removed implfree presolver
    - See also the full release notes and the release paper.

3.17.1.5.16 SELKIE  The SELKIE solver - new to GAMS with this release - is an EMP solver that implements decomposition methods for multi-agent equilibrium and related models.

3.17.1.5.17 SoPlex

- New libraries 4.0.1.
  - new simplifier step to perform variable aggregation for equations with two variables
  - new parameter bool:ensureray to re-solve the original problem in case of infeasibility/unboundedness to get a valid proof/ray
  - See also the full release notes and the release paper.
3.17.1.5.18 Xpress

• New libraries 33.01.10.
  – local symmetry re-detection in MIP tree search
  – improved branching for highly symmetric MIPs
  – new or improved cuts, degeneracy handling, crossover, preprocessing
  – FICO does not release patches for all platforms with each patch release, so some platforms lag behind the most-used ones.

3.17.1.6 Tools

3.17.1.6.1 CSDP

• See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

3.17.1.6.2 CSV2GDX

• Extended recognition of special values:
  – N/A is recognized as NA,
  – TRUE is recognized as 1,
  – FALSE, NONE and NULL are recognized as 0.

• Added new option text to specify the column to get explanatory text from when reading a set.

• Changed handling of bad UELs: Reading a bad UEL (e.g. a UEL exceeding the maximum length) causes an error now. The old behavior can be enabled again using the new option acceptBadUels.

3.17.1.6.3 GAMS Studio

• New version 0.10.3
  – Stability and performance improvements.
  – Added a new section Comparing GAMS Studio and GAMSIDE to the documentation.
  – Added a Reference File Viewer to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS models and is created using the reference parameter when running the model.
  – Added a Tab Browser to list and search through all open files. It is accessible with a button next to the main tab bar or via the shortcut Ctrl+K.
  – GDV Viewer :
    * Added a first, basic version of a Table View : The new button ”Table View” switches the view to a table based representation. The button is only enabled for multidimensional symbols and puts the last dimension into the header of the table. Drag and drop of items is not yet supported but will be in the future.
    * Added option to hide specific value columns for variables and equations.
    * Added support for the use of different encodings using the ”Edit->Encoding” menu. The default encoding for a GDX file generated in Studio by running a GMS file with automatic GDX creation (F10) is the one used by the GMS file.
  – Editor
* Added smart typing: Automatic insertion of closing character for brackets and quotes.
* Added ability to select text and press either a opening bracket or quote to surround text with these symbols.
* Improved backspace behavior, if auto indentation is active.

- Output Widget
  * Added info, warning, and error messages to GAMS Studio system log.
  * GAMS log files are now written to disk (can be disabled in settings).
  * Added "clear log" context menu entry.

- Project Explorer
  * Studio automatically adds all files specified through GAMS parameters (.gdx, .ref, .lst) to the Project Explorer.
  * Allow to rename Project Explorer groups.
  * Groups are closed if they are empty.
  * If a group is closed the associated GAMS process will be stopped.
  * Added drag and drop of files between groups.
  * Added shortcut to focus Project Explorer (Ctrl+J) and allow to navigate using arrow keys to select a file.
  * Added "Select all", "Expand all" and "Collapse all" to the context menu.
  * Added shortcut to select all files (Ctrl+A).

- Various
  * Added context menu for tabs (offering different close actions).
  * Added animation for groups with running GAMS processes.
  * Studio comes to foreground when double clicking a Studio-associated file.
  * The "About GAMS" dialog contains many important information like used GAMS version, system directory, or license information. It features a "Copy System Information" button which copies all of these information to easily attach them to a support request. The dialog has a second tab which gives an overview about solvers, their license status, and what kinds of capabilities they have.
  * "File -> New..." dialog starts with a default for a new file name.
  * "File -> Open..." (or drag'n'drop) of one file now focuses that file if it is already present in the Project Explorer.
  * New entry "File -> Open... in new group" to force creation of new group (even if the is already present in the Project Explorer).
  * "File -> Open..." (or drag'n'drop) of multiple files puts all files opened at once into a single group instead of individuals.
  * Added shortcut to focus command line parameter edit field (Ctrl+L).
  * Added Ctrl+F as close shortcut for search dialog.
  * Changed shortcut to open Settings dialog to F7.

Note
This version of GAMS Studio does not support the Dark Mode of MacOS Mojave. Please disable this mode to use GAMS Studio.

3.17.1.6.4 GDXDUMP

- New options for writing out special values: NAOut, UndfOut, PINfOut, MINfOut, and ZeroOut.
- Fixed broken generated GAMS code, if UelTable option was used, but there were no UELs.
- Lower and upper bounds of equations are dumped always now (even if their values are the default). This fixes a potential problem, that the equation type is unknown for the generated GAMS code.
- When writing CSV files the element text of set elements could not be written. Now with enabled option CSVSetText the element text of a set element is written as the last column in the CSV file.
3.17.1.6.5 GDXXRW

- Fixed a bug which could have caused a crash when reading merged cells from Excel using the GDXXRW options useRC and cMerge=1.

3.17.1.6.6 MODEL2TEX

- The new script `model2tex.sh` sets the required dependencies for the model2tex tool on Linux and Mac OS X. Using this script is the recommended way for using model2tex. Calling the Python script directly is not recommended anymore since the used Python interpreter would need to be configured manually. `model2tex.sh` uses the Python interpreter that comes with the GAMS distribution.

3.17.1.7 Object Oriented APIs

3.17.1.7.1 Python

- The `setup.py` script now installs the idx expert-level API.

3.17.1.8 Model Libraries

3.17.1.8.1 GAMS Data Library  New models added:

- `csv2gdx2.gms` : CSV2GDX Example 2 - Reading CSV Files with CSV2GDX (111)
- `csv2gdx3.gms` : CSV2GDX Example 3 - Reading Semicolon separated Data (112)
- `csv2gdx4.gms` : CSV2GDX Example 4 - Dealing with missing Labels and Duplicates (113)
- `csv2gdx5.gms` : CSV2GDX Example 5 - Reading more than one Parameter from a single Input File (114)
- `csv2gdx6.gms` : CSV2GDX Example 6 - Reading economic Data from the World Bank Data Catalog (115)
- `csv2gdx7.gms` : CSV2GDX Example 7 - Reading special Values with CSV2GDX (116)
- `csv2gdx8.gms` : CSV2GDX Example 8 - Reading an compressed encrypted Input File (117)
- `csv2gdx9.gms` : CSV2GDX Example 9 - Reading Options from an external File (118)
- `GDXXRWExample17.gms` : Reading several Scalars from Spreadsheet (119)
- `GDXXRWExample18.gms` : Reading Sets from Spreadsheet (120)
- `GDXXRWExample19.gms` : Writing Parameter to Spreadsheet including Zero Values (121)
- `GDXXRWExample20.gms` : Reading empty Cells with colMerge and reading merged Excel Ranges with cMerge (122)
- `GDXXRWExample21.gms` : Skipping empty Rows or Columns and Ignoring Rows or Columns (123)
- `GDXDUMPExample1.gms` : GDXDUMP - Adding double Quotes to an user defined Header when writing to CSV (124)
3.17.1.8.2 GAMS EMP Library  New model added:

- `emppython1.gms`: Three Simple EMP Models with Indexed EMP Syntax and Python Parser (104)

3.17.1.8.3 GAMS Test Library  New models added:

- `csv2gdx7.gms`: CSV2GDX - Checking the Error Messages for incorrect Parameter Input (752)
- `csv2gdx8.gms`: CSV2GDX - Testing the `valueDim` Option (753)
- `csv2gdx9.gms`: CSV2GDX - Testing the Field Separator Tab (754)
- `csv2gdx10.gms`: CSV2GDX - Testing the Field Separator Semicolon (755)
- `csv2gdx11.gms`: CSV2GDX - Testing the Field Separator Comma (756)
- `selkie01.gms`: SELKIE test suite (757)
- `selkie02.gms`: SELKIE test suite (758)
- `selkie03.gms`: SELKIE test suite (759)
- `selkie04.gms`: SELKIE test suite (760)
- `selkie05.gms`: SELKIE test suite (761)
- `selkie06.gms`: SELKIE test suite (762)
- `selkie07.gms`: SELKIE test suite (763)
- `selkie08.gms`: SELKIE test suite (764)
- `selkie09.gms`: SELKIE test suite (765)
- `selkie10.gms`: SELKIE test suite (766)
- `selkie11.gms`: SELKIE test suite (767)
- `selkie12.gms`: SELKIE test suite: sub-diagonalization (768)
- `selkie13.gms`: SELKIE test suite: dualvar (769)
- `selkie14.gms`: SELKIE test suite: isolated implicit variable (770)
- `selkie15.gms`: SELKIE test suite: proximal perturbation (771)
- `selkie16.gms`: SELKIE test suite: obj variable setting (772)
- `selkie17.gms`: SELKIE test suite: obj variable setting (773)
- `selkie18.gms`: SELKIE test suite: obj variable not free (774)
- `selkie19.gms`: SELKIE test suite: equation marginal values (775)
- `put12.gms`: Testing `put` utility solver (776)
- `gdxxrw8.gms`: GDXXRW - Testing the Option intAsText (777)
- `gdxxrw9.gms`: GDXXRW - Testing the Option checkDate (778)
- `gdxxrw10.gms`: GDXXRW - Testing cMerge when reading Sets with the values Option (779)
- `gdxxrw11.gms`: GDXXRW - Reading and writing special Values (780)
- gdxrwl2.gms: GDXXRW - Testing the skipEmpty and cMerge Option
- gdxrwl3.gms: GDXXRW - Testing the values Option when reading or writing Set Elements
- gdxrwl4.gms: GDXXRW - Testing different Excel Range Specifications
- gdxdump2.gms: GDXDUMP - Testing the dumping Functionality on several GDX Files
- gdxdump3.gms: GDXDUMP - Dumping special Values of a Parameter from GDX
- gdxdump4.gms: GDXDUMP - Dumping special Values of Variable-Subfields from GDX
- onmulti8.gms: Test for $onMultiR
- implset1.gms: Test for Implicit Set Definition
- emppy1.gms: Test an equilibrium model using emp python
- emppy2.gms: Formulate the simplevi.gms example using emp python
- emppy3.gms: Test a combination of optimization and vi agents using emp python
- miqcp04.gms: Test behavior for integer infeasible model
- duplic01.gms: Detecting duplicate entries in unsorted data

3.17.1.9 Solver/Platform availability

<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON 18.5.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.8</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.6</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 8.1</td>
<td>7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.18 25.1 Distribution

#### 3.18.1 25.1.1 Major release (May 19, 2018)

#### 3.18.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Eden Huang, Katja Buhrkal Jensen, Jan-Erik Justkowiak, Erwin Kalvelagen, Marko Loparic, Bruce McCarl, Scott McDonald, Edmund Moshammer, Andres Ramos, Tom Rutherford, Wilfredo Sifuentes, and Tapio Westerlund.

#### 3.18.1.2 Platforms

- On Windows, some solvers in the GAMS distribution as well as GAMS Studio have dependencies on certain MS Visual C++ runtime libraries. Most of them are included in the GAMS distribution or are present on most Windows systems. However, we are aware that in rare cases, some libraries are missing. In such situations, we advise to run the appropriate installer for these libraries:
  - On a 32-bit GAMS version, execute `[GAMS system]\apifiles\C++\lib\vs2013\vcredist_x86.exe`, followed by `[GAMS system]\apifiles\C++\lib\vs2015\vcredist_x86.exe`.
  - On a 64-bit GAMS version, execute `[GAMS system]\apifiles\C++\lib\vs2017\vcredist_x64.exe`.  

<table>
<thead>
<tr>
<th>Solver</th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 11.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 12.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 8.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OQNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 6.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 4.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 33.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

32.01 29.01
3.18.1.3 GAMS System

3.18.1.3.1 GAMS

- New conditional expressions to be used in conditional compilation, namely `gdxDimension`, `gdxEquType`, `gdxParType`, `gdxSetType`, `gdxSymExist`, `gdxVarType`, and `uelExist`. Details about these can be found [here](#).

- New command line option `LstTitleLeftAligned`: Write title of LST file left aligned.

- New command line option `ShowOSMemory`: Show the memory usage reported by the operating system (either resident set size or virtual set size) instead of the internal counting.

- GAMS throws now a compilation error if an unexpected suffix at a symbol is encountered in an option statement involving identifiers. In the past such suffixes were ignored, which could have given the impression that they were actually doing something. So, all the `.dim1` suffixes in the following example will create an error now:

```plaintext
Set i(*)
   j(*)
   ij(*,*) / i1.j1, i1.j2, i2.j1, i2.j2 /

Option i < ij.dim1;
Option i <= ij.dim1;

Set k / k1*k6 /;
Parameter ii(*) / i1 4, i2 5, i3 6 /
   iii(k,*)

Option iii > ii.dim1;
Option shuffle = ii.dim1;
```

- The details of the expansion of multi-line macros were changed with this release. In the past, trailing white space in the macro definition was always removed. Now, in multi-line macros, the white space before the continuation character `\` is preserved. This example demonstrates the difference:

```plaintext
$macro xAndY(x,y) x and \
   y
scalar x /1/;
if(xAndY(1,x), display 'true');
```

In the past, this caused a compilation error, since the macro was expanded to `if(1 and y, display 'true');` ("and" and "y" were concatenated to "andy"). Now that the white space after "and" is preserved, this does not cause a problem anymore.

- The statement `Alias (k,k);` for an unknown symbol `k` causes a compilation error now.

- In rare cases, embedded code recognized indented Python code incorrectly. This has been fixed.

- New property `gams.debug` for controlling debug information of the Object-oriented API in embedded code sections.

- Fixed a problem related to SSL support in `GMSPython` that prevented certain tools (e.g. pip) from working on Linux and Mac OS X.

- Models of type MCP can now be solved with modifiable parameters in the GUSS framework and as OO-API `GAMSModelInstance`. 
3.18.1.4 Solvers

3.18.1.4.1 BARON

- New libraries 18.5.8.
  - Updated to Ipopt 3.12.8 and CBC 2.9.9.
  - Replaced Ipopt's linear solver MUMPS by MA57 from HSL, a collection of Fortran codes for large-scale scientific computation. Thus, when using IPOPT as NLP solver, BARON may now be more robust and a little faster, especially for problems without integer variables.
  - New bounds reduction strategies based on optimality conditions can reduce the size of the branch-and-bound tree.
  - Rewrote core memory management routines for speed increase and memory reduction.
  - On some problems, BARON currently provides incorrect marginal values. This problem will be fixed in the near future.

- The use of Ipopt in BARON on MacOS X is no longer disabled by default.

- There will be no future updates of BARON for Windows 32-bit.

3.18.1.4.2 Bonmin(H), CBC, Couenne

- Wall-clock time is now always used to apply a timelimit to Branch-and-Bound. The nww option clocktype can be used to switch back to CPU time.

3.18.1.4.3 CONOPT

- New libraries 4.05 for CONOPT 4.
  - Improvements for multi-threading.

- New libraries for CONOPT 3.

- Fixed problems with redundancy in preprocessor and an issue related to system error 2024.

3.18.1.4.4 ConvertD

- Added options PermuteVars and PermuteEqus to permute the declaration of variables and equations, respectively, in scalar GAMS output.

- Tweaked printing of variable bounds and activity levels in scalar GAMS output.

- Fixed handling of infinite upper bounds for integer variables in scalar GAMS output: A line that sets GAMS option intvarup to 0 is now added to the output.

3.18.1.4.5 Couenne

- New libraries.
3.18.1.4.6 DICOPT

- Improvements to Feasibility Pump:
  - Fixed handling of nonlinear objective function when maximizing.
  - Fixed update of cutoff decrement after a solution with objective value very close to zero has been found.
  - Fixed translation of NLP projection problem optimal value to norm of associated projection. Changed default for option `fpprojzerotol` from 1e-6 to 1e-4.
  - Added creation of cuts derived from solution of NLP projection problem. Use option `fpprojcuts` to disable.
  - Added option `fpintegercuts` to disable integer cuts or enable them for mixed-binary problems only. The latter is the new default.
  - Added option `fp_mipgap` to specify an optimality tolerance (relative gap limit) for the MIP projection problem.
  - The `stall limit` is now only applied after a first solution has been found. Changed default for option `fpiterlimit` to 20.

3.18.1.4.7 GUROBI

- New libraries 8.0.
  - As announced for GAMS 24.9, Gurobi 8 is no longer supported for Windows 32-bit. We continue to ship Gurobi 7.5 for this platform.
- New partition heuristic based on user annotation via the new `dot` option `.partition` and enabled with new option `PartitionPlace`.
- Completely new interface to Gurobi's Remote Services (i.e. compute server, distributed algorithm, and instant cloud). For details see Compute Server and the following sections.

3.18.1.4.8 JAMS

- Added capability to handle QVI models.
- The options controlling the reformulation strategy for shared variables in multi-agent EMP models have changed: see ImplVarModel for details.

3.18.1.4.9 LocalSolver

- New libraries 7.5 (20180405).
- To use LocalSolver 7.5, a machine-specific LocalSolver license is now required on Linux, too. See the solver manual on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

3.18.1.4.10 Mosek

- New libraries 8.1.0.53.
3.18.1.4.11 ODHCPLEX

- A new solver GAMS/ODHCPLEX from Optimization Direct Inc. has been added to the GAMS solver portfolio.
- The solver implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models and uses IBM CPLEX as its underlying solver engine.
- The heuristics decomposition method works in an automatic fashion or can be guided by user specified selections.
- Users will need a GAMS/CPLEX or GAMS/Cplex link license for this solver to work. Commercial users will also need a GAMS/ODHCPLEX license.
- Currently, the solver is available for Windows 64-bit only.

3.18.1.4.12 SCIP

- New libraries 5.0 (09c736f).
  - changed default for parameter constraints/quadratic/gaugecuts from 1 to 0
  - changed default for parameter heuristics/completesol/freq from 1 to 0
  - changed default for parameter separating/cmir/freq from -1 to 10
  - changed default for parameter separating/cmir/freq from 1 to 0
  - changed default for parameter separating/flowcover/freq from -1 to 10
  - changed default for parameter separating/flowcover/freq from 1 to 0

3.18.1.4.13 Xpress

- New libraries 32.01.10 containing several minor bug fixes and performance improvements.

3.18.1.5 Tools

3.18.1.5.1 GAMS Studio

- This release contains the first preview version of GAMS Studio - a completely new integrated development environment for GAMS. GAMS Studio is still in a very early stage of development, but we believe it is already mature enough to be a productive tool. GAMS Studio is available for Linux, Mac OS X, and Windows only.
- GAMS Studio combines many (but not all) features of the classic GAMS IDE with some new elements:
  - a code editor to write GAMS models (including basic syntax coloring, block edit mode, etc.),
  - execution of GAMS models,
  - an output panel that presents the progressing GAMS log,
  - a listing viewer (including a tree view to navigate through the file) to inspect the listing file,
  - an interactive option editor to set GAMS parameters,
  - a GDX viewer that shows the table of content and data of GDX files and offers useful details like sorting and filtering of data,
– a project explorer helps to manage different projects in one session, and
– an integrated help system to make it easier to find additional information, e.g., by pressing F1
  while the cursor is on a GAMS keyword in the code editor.

• We encourage our users to weigh in, help prioritize GAMS Studio's future development directions,
  or provide any other feedback. If you find problems, miss features, or have comments, please send a
  note to studio@gams.com.

• It is also possible to contribute directly or build on top of GAMS Studio, since the source code is
  available on GitHub (https://github.com/GAMS-dev/studio) under the GNU GPL license (see

• We plan to provide updates for GAMS Studio frequently in the near future, also separate from new
  releases of the GAMS distribution. To facilitate the update process, Studio includes an interactive
  check (requiring an Internet connection) for the availability of newer versions. This "Check for
  Update" button can be found below the menu "Help".

• A short overview about GAMS Studio can be found in the YouTube channel GAMS Lessons.

  Note

  – New options to define the association for GAMS files (.gms) have been added to the
    installer for GAMS on Windows. For now, the GAMS IDE is still the default choice, but
    this may change for future releases.
  – On Linux, Studio is installed in the form of an AppImage as part of the GAMS system
    directory.
  – On Mac OS X, Studio is part of the DMG installer only. It is installed as an additional
    application in the Applications directory next to the GAMS Terminal application and is
    available through the Launchpad. GAMS files (.gms) are associated with GAMS Studio.

  Attention

  – In rare cases, GAMS Studio crashes when the Open or Save dialog gets opened. This is
    mostly related to third party software, e.g. older versions of "Dell Backup and Recovery".
    It is recommend to update, switch off or uninstall the software.
  – We are aware of some performance issues at this stage of development. So while things
    work well, for example, with the models from our model libraries, you might experience
    some delay when working with huge files, e.g., while searching or when the log is processed
    for very long output.

3.18.1.5.2 XLSDump

• Sometimes, Excel Spreadsheets contain links to pictures, which cannot be found. This situation
  caused an error in XLSDump in previous versions. Now, such an error is ignored.

3.18.1.6 Object Oriented APIs

• New implementation of GAMSJob.Interrupt() on Linux and Mac OS X that removes dependency
to the command line tool pstree.

3.18.1.6.1 C++

• Added support for Microsoft Visual Studio 2015 on Windows 32-bit.
3.18.1.6.2 Java

- Changed the naming scheme of a temporary working directory to be created from `yyyyMMddHHmmss` to the prefixed `gams` (defined by `GAMSGlobals.workingDirectoryPrefix`), in case no working directory has been specified.

- Fixed the behavior when a GAMSDatabase is added with a name that already exists. A GAMSException will be raised now (see `GAMSWorkspace.addDatabase(String databaseName)` and `GAMSWorkspace.addDatabaseFromGDX(String gdxFileName, String databaseName)`).

- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available, because calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially create unpredictable outcome. Whenever the object is no longer needed it is recommended to explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See `GAMSDatabase.dispose`, `GAMSModelInstance.dispose`, and `GAMSOptions.dispose`.

3.18.1.7 Expert Level APIs

- All Java native interfaces to `expert-level APIs` are now included in `\[Path/To/GAMS]/apifiles/Java/api/GAMSJavaAPI.jar`.

3.18.1.7.1 GMO

- The constant `MAXEVALTHREADS` (i.e. the number of parallel threads supported for function and derivative evaluations in the solver interface library GMO) has been increased from 16 to 64. This effectively increases the same limit in the CONOPT4 solver.

3.18.1.8 Model Libraries

3.18.1.8.1 PSOPTLIB - Power System Optimization Modelling in GAMS

- This new library by Alireza Soroudi has been added to the GAMS system. This is a collection of the models based on the book *Power System Optimization Modelling in GAMS* by Alireza Soroudi. The library contains a selection of 32 models from various areas of power system optimization expressed in GAMS. Book and library describe how the General Algebraic Modeling System (GAMS) can be used to solve various power system operation and planning optimization problems. The book is the first of its kind to provide readers with a comprehensive reference that includes the solution codes for basic/advanced power system optimization problems in GAMS, a computationally efficient tool for analyzing optimization problems in power and energy systems. The book covers theoretical background as well as the application examples and test case studies. It is a suitable reference for dedicated and general audiences including power system professionals as well as researchers and developers from the energy sector and electrical power engineering community and will be helpful to undergraduate and graduate students.

- You can retrieve the individual models through the IDE and Studio model library browser, via the command line utility `psoptlib` or through calls in the Object Oriented APIs.

3.18.1.8.2 GAMS EMP Library

- `simpleqvi1.gms` : Simple Quasi-Variational Inequality (101)
- `simpleqvi2.gms` : Simple Quasi-Variational Inequality (102)
- `simpequil3.gms` : Simple Generalized Nash Equilibrium Problem (103)
3.18.1.8.3 GAMS Model Library

- guss2dim.gms: Two dimensional scenario GUSS Example (423)
- obstacle.gms: An Obstacle Problem (424)
- csched: Added two more formulations for a related problem. Contributed by Tapio Westerlund.

3.18.1.9 Solver/Platform availability

<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.8</td>
<td>12.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>12.6</td>
<td>✓</td>
</tr>
<tr>
<td>DECI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 10.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALSOLVER 7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLINP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPiece</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OQNLP</td>
<td>✓</td>
<td>32bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.18.2 25.1.2 Minor release (August 01, 2018)

#### 3.18.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Shaoyan Guo, Eden Huang, David Laudy, Evangelos Panos, Wilfredo Sifuentes and Larissa de Vries.

#### 3.18.2.2 Platforms

- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

#### 3.18.2.3 GAMS System

##### 3.18.2.3.1 GAMS

- Fixed a bug which caused an unexpected error when using the + operator in a model statement with the following symbol being a model itself, like this:

```
variable x;
equation e1, e2;
e1.. x =L= 5;
e2.. x =G= 3;
model m2 /e2/;
model m /e1+m2/;
solve m min x use lp;
```

#### 3.18.2.4 Solvers

##### 3.18.2.4.1 BARON

- Fixed translation of branching priority values when specified in BARON options file and GAMS option PriorOpt not enabled.
- Specifying maxdouble as branching priority in a BARON options file now behaves as if specifying inf as branching priority in a GAMS model.
3.18.2.4.2 BONMIN

- The default for option `number_cpx_threads` is now initialized according to the value of the GAMS option `threads`.

3.18.2.4.3 CONOPT

- New libraries 3.17I for CONOPT 3.
- New libraries 4.06 for CONOPT 4.

3.18.2.4.4 GUSS

- Models with variable parameters and model attribute `holdFixed=1` could not be solved with 25.1.1. In this release the `holdFixed` option is automatically turned off.

3.18.2.4.5 Ipopt

- New libraries.
  - Fixed problems in handling of time limit (reslim).

3.18.2.4.6 LocalSolver

- New libraries 7.5 (20180601).

3.18.2.4.7 ODHCPLEX

- New libraries 3.4.3.
- ODHCPLEX is now available for Linux, too.

3.18.2.4.8 SoPlex

- New libraries 3.1 (b0e0048).

3.18.2.4.9 SCIP

- New libraries 5.0 (1d9c207).

3.18.2.5 Tools

3.18.2.5.1 MODEL2TEX

- Backslashes in the explanatory text of symbols are replaced automatically with `\textbackslash`. 
3.18 25.1 Distribution

3.18.2.5.2 GAMS Studio

- New version 0.9.2
  - Stability improvements
  - Allow only one instance of Studio to run at the same time (when file association for Studio is active, a double click on a gms file will open that file in a running Studio instance and does not open a new Studio)
  - Added startup parameter \texttt{--gams-dir} to specify a non-default GAMS system folder to be used
  - Added shortcuts for Interrupt and Stop (F12 and Shift+F12)
  - Restructured the File -> Encoding menu
  - About dialog: added button to copy product information to clipboard
  - GDX Viewer: added facility to search for symbols
  - Option Editor: allow to use F1 to open help
  - Project Explorer:
    - "Add Existing Files" allows to add multiple files at once
    - Added context menu entry to open the log for a group-node
  
  \textbf{Note}
  - When migrating from an older version of GAMS Studio, one might have an empty project explorer and no recent files listed. This is expected behavior because of a bug fix within the GAMS Studio settings file.

3.18.2.6 Object Oriented APIs

3.18.2.6.1 Python

- Fixed a bug that prevented the setup.py file to be called with parameter \texttt{--noCheck} for turning off the version check.
- The setup.py script can be used from an arbitrary location. It is not required anymore to switch to the files location before installing.

3.18.3 25.1.3 Minor release (October 30, 2018)

3.18.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release.

3.18.3.2 GAMS System

3.18.3.2.1 GAMS

- Fixed a bug which caused the \texttt{sysIdent} property not being set correctly after a solve statement.
- Fixed a bug which could have caused a problem with \texttt{execute_load} being called after \texttt{endEmbeddedCode} in a loop.
3.18.3.3 Solvers

3.18.3.3.1 ANTIGONE

- If interrupting ANTIGONE (using Ctrl+C, for example), the final trydual NLP solve does no longer stop immediately in the first iteration.

3.18.3.3.2 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.4 on Mac OS X.

3.18.3.3.3 CONVERTD

- When using options Jacobian or Hessian, new symbols iobj, objConst, and objJacVal are written to the GDX file. If the model can be reformulated with a true objection function, the symbol iobj contains the label of the objective defining equation, while the symbols objConst and objJacVal constain the objective constant and the coefficient of the objective variable. If the model cannot be reformulated with a true objective, iobj will be empty and objConst and objJacVal will be 0.

3.18.3.3.4 GUROBI

- New libraries 8.1.0.
- Option GUROBi preqlinearize allows the value of 2 (Force Linearization and get compact relaxation).

3.18.3.3.5 JAMS

- Added some experimental indexed EMP syntax. See example model emppython1.gms: Three Simple EMP Models with Indexed EMP Syntax and Python Parser.

3.18.3.3.6 Lindo/LindoGlobal

- New libraries 11.0.338.

3.18.3.3.7 MOSEK

- New libraries 8.1.0.64.

3.18.3.4 Tools

3.18.3.4.1 CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.
3.18.3.5 Object Oriented APIs

3.18.3.5.1 Python

- The UpdateAction Primal and Dual in a GamsModifier were not recognized and triggered an exception. This has been fixed.

3.18.3.6 Model Libraries

3.18.3.6.1 GAMS EMP Library

- emppython1.gms: Three Simple EMP Models with Indexed EMP Syntax and Python Parser (104)

3.19 25.0 Distribution

3.19.1 25.0.1 Major release (January 17, 2018)

3.19.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Jeff Dischler, Michael Ferris, Dhruv Gupta, Erwin Kalvelagen, Florian Landis, Andre Lemelin, Erfan Mohagheghi, Anthony Paul, and Nathan Sudermann-Merx.

3.19.1.2 Platforms

- Due to the discontinuation of Cplex on Windows 32bit (with Cplex 12.7 in 2016) we are faced with potentially phasing out/dropping solvers depending on Cplex on Windows 32bit with the next major release. The most likely candidate for this is ANTIGONE. These solvers will still be available for 64-bit Windows and other supported platforms. Other solver vendors (e.g. Gurobi and FICO-Xpress) have also announced the discontinuation of their product on the Windows 32bit platform. If you rely heavily on the availability and support of the Windows 32bit platform please contact support@gams.com to discuss your options.

- As announced, dropped support for x86-64 Solaris.

- As announced, increased the minimal required MacOS X version to 10.11.

- As announced, increased the minimal required GLIBC version on Linux to 2.12.
3.19.1.3 GAMS System

3.19.1.3.1 GAMS

- Extended the $offEmbeddedCode facility to allow the use of a "projection operator":
  $offEmbeddedCode {symbol[[<]=]embSymbol[.dimX]]}
  More information can be found here.
- Added support for the Object-oriented GAMS Python API in the Embedded Code Facility for Python. The method gams.set() accepts instances of type GamsParameter, GamsSet, etc. as data. Instances of GamsWorkspace and GamsDatabase can be retrieved using the properties gams.ws and gams.db. The property gams.wsWorkingDir can be used to specify the working directory of the created GamsWorkspace.
- Added $libinclude pyEmbMI to conveniently work with Python OO-API GamsModelInstance from embedded code. See embmiex1.gms: Simple Embedded Code ModelInstance example.
- Fixed a problem in embedded code when reading an empty scalar symbol.
- Fixed a problem when solving a model with solveLink 5, 6, or 7 with communicating scales without scaleOpt active.
- The EPS value in the Embedded Code Facility for Python has been changed to 4.94066E-324. This is the same value that is used for EPS in the Object-oriented Python API.
- New Put Utility save: Writes a save file of the current state of execution.
- Improvement for the Put Utility Statement: It is no longer required to define a file and activate it, just to use a put utility.
- New command line option fileStemApFromEnv: Append a string read from an environment variable to the fileStem.
- Change to command line option fileStem: Create an immediate error when the value contains a "\" or "/" character to avoid problems later on.
- Fixed a bug with execute unload, which could have caused a set to be exported mistakenly as alias to a different set, if symbol renaming was used.
- Fixed a bug which prevented to open more than 65,000 put files.
- Solves with scaleOpt=1 using GUSS or instantiations of OO-API GAMSModelInstances resulted in some circumstances in the report of scaled solutions or other erroneous behavior. Hence GAMS will now reset the modelname.scaleOpt attribute to NA before such a solve or GAMSModelInstance generation. As a consequence, models solved via GUSS or as a GAMSModelInstance in the OO-APIs will not be scaled by GAMS.
- Solving MCP models with scaleOpt=1 using the solvers AMPL or PYOMO creates an execution error now. In the past this lead to wrong results potentially.

3.19.1.4 Solvers

3.19.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.0 for Linux and Mac OS X and to version 2018.1 on Windows 64-bit.

3.19.1.4.2 CBC

- New libraries.
3.19.1.4.3 CONOPT

- New libraries 3.17G (Conopt3).
- New libraries 4.04 (Conopt4).
  - Fixed a system error 65666.

3.19.1.4.4 Couenne

- New libraries.
  - Bugfixes and tuning for fixed point bound tightening.
- If Couenne does not find a feasible solution, but the initial point is feasible, then the initial point is now reported back to GAMS.

3.19.1.4.5 CONVERT

- If the filename provided for the Jacobian or Hessian option contains the string novenames ConvertD will not export the original equation and variable names as set text to elements of sets i and j. Having the original equation and variable names can make the GDX file significantly larger and slow to write and read.
- Fixed a problem when trying to write scaled MCP models.

3.19.1.4.6 CPLEX

- New libraries 12.8.0.
- New parameters
  - SubMIPScale: Parameter to scale the problem matrix when CPLEX solves a subMIP during MIP optimization
  - SubMIPStartAlg: Starting algorithm for subMIP of a MIP
  - SubMIPSubAlg: Algorithm for subproblems of a subMIP of a MIP
  - DynamicRows: Switch for dynamic management of rows
  - Sifting: Switch for sifting from simplex optimization
- In CplexD allow the BendersPartition to be set via the .stage variable suffix together with option BendersPartitionInStage.

3.19.1.4.7 Gurobi

- Fixed a problem for model with SOS variables but no constraints.

3.19.1.4.8 KESTREL

- New options neos_username and neos_user_password can be used in the option file in order to submit authenticated jobs using a NEOS user account.
3.19.1.4.9  LocalSolver

- New libraries 7.5 (20171117) for Mac OS X and Windows.
  - Preprocessing entirely rewritten: size reduction by a factor up to 10 on some huge instances.
  - Combinatorial models based on booleans and integers: performance improvement and increased ability to prove optimality.
  - Continuous linear and nonlinear models: performance improvement through the integration of state-of-the-art algorithms.
- Note, that to use LocalSolver 7.5, a machine-specific LocalSolver license is now required. See the solver manual on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

3.19.1.4.10  Mosek

- New libraries 8.1.0.34.

3.19.1.4.11  SBB

- Fixed a bug introduced in 24.8 that prevents the infeasSeq option to work.

3.19.1.4.12  SCIP

- New libraries 5.0 (2b35b18).
  - SCIP can now automatically detect and exploit symmetries in MIPs (Linux and Mac OS X only)
    * added parameter misc/usesymmetry to determine which symmetry handling should be used
    * added parameter groups presolving/symmetry, presolving/symbreak, constraints/symresack,
      constraints/orisack, constraints/orbitope
  - Presolving
    * new presolver sparsify that tries to cancel nonzero coefficients in linear constraints by adding multiples of linear equalities
    * disabled reformulation of products of a binary variable with a linear term that does not solely involve binary variables
      · added parameter constraints/quadratic/binreformbinaryonly to disable reformulation of products of binary and non-binary variables
    * revised disaggregation of quadratic constraints: the number of created constraints can now be controlled and the disaggregated constraints are scaled in order to increase numerical accuracy
      · replaced constraints/quadratic/disaggregate by constraints/quadratic/maxdisaggrsize to bound the total number of created constraints when disaggregating a quadratic constraint
      · added parameter constraints/quadratic/disaggmergemethod to change the strategy of how to merge independent blocks of quadratic constraints
  - Primal Heuristics
    * new primal heuristic ALNS that orchestrates eight different LNS heuristics adaptively using algorithms for the multi-armed bandit problem
* new primal heuristic MPEC that solves a MPEC reformulation of a mixed-binary nonlinear problem by regularized NLP reformulations
* improved the clique and variable bound pre-root heuristics, which are now often able to fix many more variables
  - removed parameters heuristics/clique/\{multiplier,initseed\}
  - replaced parameter heuristics/\{clique,vbounds\}/minfixingrate by heuristics/clique/minintfixingrate, heuristics/vbounds/minintfixingrate, heuristics/clique/minmipfixingrate, and heuristics/vbounds/minmipfixingrate, which check the fixing rate before LP solving and after sub-MIP presolve
  - added parameters heuristics/clique/maxbacktracks and heuristics/vbounds/maxbacktracks to limit the number of backtracks in the fix-and-propagate phase
  - added parameters heuristics/clique/uselockfixings and heuristics/vbounds/uselockfixings to enable fixing of additional variables based on variable locks
  - added parameters heuristics/vbounds/feasvariant and heuristics/vbounds/tightenvariant to specify the fixing variants used by the vbounds heuristic
  - changed default for parameters heuristics/clique/freq and heuristics/vbounds/freq from -1 to 0
  - changed default for parameter heuristics/clique/priority from -1000500 to 5000
  - changed default for parameter heuristics/vbounds/priority from -1100600 to 2501
* added parameters heuristics/completesol/beforepresol, heuristics/completesol/maxlpiter, and heuristics/completesol/maxcontvars
* changed default for parameter heuristics/indicator/oneopt from 1 to 0
* changed default for parameter heuristics/locks/minfixingrate from 0.25 to 0.65
* changed default for parameter heuristics/locks/priority from 2000 to 3000

- Separation (cutting planes)
  * utilizing linear inequalities (computed in the OBBT propagator) to compute stronger linearizations for bilinear terms
    - added parameters constraints/quadratic/usebilinameqbranch, constraints/quadratic/minscorebilinterms, and constraints/quadratic/bilinameqmaxseparounds
    - added parameter constraints/quadratic/mincurvcollectbilinterms to change the minimal curvature of constraints to be considered when returning bilinear terms to other plugins
    - added parameters propagating/obbt/itlimitfactorbilin, propagating/obbt/minnonconvexity, and propagating/obbt/createbilinameqs
  * improved cut post-processing (apply coefficient tightening, enforce maximal dynamism), selection, and management
    - added parameters separating/maxlocalbounddist, separating/maxcoefratio, and separating/intsupportfac
    - removed parameter separating/orthofac
    - changed default for parameter separating/cutagelimit from 100 to 80
    - changed default for parameter separating/minefficacy from 0.05 to 0.0001
    - changed default for parameter separating/minefficacyroot from 0.001 to 0.0001
    - changed defaults for parameters separating/minortho and separating/minorthoroot from 0.5 to 0.9
    - changed default for parameter separating/objparalfac from 0.0001 to 0.1
    - changed default for parameter separating/poolfreq from 0 to 10
  * MIP cutting planes are now separated within the tree search, too
    - parameter separating/maxstallrounds only applies to nodes in the tree (not the root node, anymore); use the new parameter separating/maxstallroundsroot for the root node
    - added parameters separating/\*/expbackoff to all separators, which increases the frequency exponentially over the depth in the tree
    - added parameter separating/maxincrounds
• changed default for parameter `separating/maxstallrounds` from 5 to 1
• changed default for parameter `separating/maxrounds` from 5 to -1
• changed default for parameter `separating/maxroundsrootsubrun` from 1 to -1
• changed default for parameter `separating/gomory/delayedcuts` from 1 to 0
• changed default for parameter `separating/gomory/freq` from 0 to 10
• changed default for parameter `separating/gomory/maxbounddist` from 0 to 1
• changed default for parameter `separating/impliedbounds/freq` from 0 to 10
• changed default for parameter `separating/impliedbounds/maxbounddist` from 0 to 1
• changed default for parameter `separating/strongcg/freq` from 0 to 10
• changed default for parameter `separating/strongcg/maxbounddist` from 0 to 1
• changed default for parameter `separating/zerohalf/freq` from -1 to 4
• changed default for parameter `separating/zerohalf/maxbounddist` from 0 to 1

* new implementation of zerohalf separator
  • added parameters `separating/zerohalf/badscore`, `separating/zerohalf/densityoffset`, `separating/zerohalf/goodscore`, `separating/zerohalf/maxcutcands`, `separating/zerohalf/maxrowdensity`, `separating/zerohalf/maxslack`, `separating/zerohalf/maxslackroot`, and `separating/zerohalf/minviol`
  • removed parameters `separating/zerohalf/delayedcuts`, `separating/zerohalf/ignoreprevzhcuts`, `separating/zerohalf/maxcutcands`, `separating/zerohalf/maxcutfound`, `separating/zerohalf/maxcutfoundroot`, `separating/zerohalf/maxdepth`, `separating/zerohalf/maxncalls`, `separating/zerohalf/maxtestdelta`, `separating/zerohalf/onlyorigrows`, `separating/zerohalf/relaxcontvars`, `separating/zerohalf/scalefraccoeffs`, `separating/zerohalf/trynegscaling`, `separating/zerohalf/usezhcutpool`, `separating/zerohalf/preprocessing/*`, and `separating/zerohalf/separating/*`
  • changed default for parameter `separating/zerohalf/maxroundsroot` from 10 to 20
  • changed default for parameter `separating/zerohalf/maxsepacuts` from 50 to 20
  • changed default for parameter `separating/zerohalf/maxsepacutsroot` from 500 to 100

* faster implementation of CMIR cut generation heuristic
  • moved many parameters for flowcover and cmir separators to new parameter group `separating/aggregation`
  • changed default for parameters `separating/cmir/freq` and `separating/flowcover/freq` from 0 to -1
  • changed default for parameters `separating/cmir/maxbounddist` and `separating/flowcover/maxbounddist` from 0 to 1
  • changed default for parameter `separating/cmir/priority` from -3000 to -100000
  • changed default for parameter `separating/flowcover/priority` from -4000 to -100000

* additional parameter changes
  • removed parameter `separating/feastolfac`
  • removed parameter `separating/cgmip/allowlocal`
  • removed parameters `separating/\{gomory,strongcg\}/maxweightrange`
  • changed default for parameter `separating/gomory/makeintegral` from 1 to 0
  • changed default for parameter `separating/gomory/maxrank` from 3 to -1
  • changed default for parameter `separating/gomory/sidetypebasis` from 0 to 1
  • added parameter `constraints/indicator/maxsepanonviolated` to stop separation after separation of non violated cuts
  • removed parameters `constraints/\{abspower,bivariate,quadratic,nonlinear\}/mincutefficacysepa`, `constraints/\{abspower,bivariate,quadratic,nonlinear\}/mincutefficacynofac`, and `constraints/soc/minefficacy`

– Propagation
  • use disjoint set to reduce peak memory usage and time to compute clique table connectedness information
  • added analysis of the clique table which identifies possible aggregations via the search for strongly connected components and may detect infeasible assignments on the way
• added parameter propagating/vbounds/minnewcliques to specify the minimum number of new cliques to trigger another clique table analysis
• added parameters propagating/vbounds/maxcliquesmedium and propagating/vbounds/maxcliquesexhaustive to limit the number of cliques relative to the number of binary variable for performing clique table analysis
• changed default for parameter propagating/vbounds/presolpriority from 0 to -90000
• changed default for parameter propagating/vbounds/presoltiming from 28 to 24
  * extended conflict analysis by analyzing dual solutions of boundexceeding LPs and improved dual ray analysis
  • removed parameters conflict/usemir and conflict/prefermir
  • added parameter conflict/sepaaltproofs
  • added parameter conflict/prefinfproof to determine whether to prefer infeasibility proof to boundexceeding proof
  • changed default for parameter conflict/useboundlp to 'b'
  • changed default for parameter conflict/maxvarsfac from 0.1 to 0.15
  • changed default for parameter conflict/maxvarsfac from 30 to 0
  – LP Relaxation
    * use LP solution polishing during probing and diving mode to activate it during many primal heuristics; remains disabled during strong branching and OBBT
    • only effective if using SoPlex as LP solver
    • added value 3 for parameter lp/solutionpolishing to enable LP polishing only during probing and diving mode
    * added parameter lp/refactorinterval to change the refactorization interval of the LP solver
  – removed parameters constraints/\{abspower,bivariate,nonlinear,quadratic,soc\}/scaling
  – added parameter groups table/* to control statistic output
  – See also the full release notes, the changelog, and the release paper.
• Fixed a problem when solving a model with solveLink 6 or 7 with scaleOpt active.

3.19.1.4.13 SolveEngine

• The GAMS time limit (reslim) is now passed to SolveEngine. Added solver option hardtimelimit to specify a time limit that is enforced on the GAMS side.

3.19.1.4.14 SoPlex

• New libraries 3.1.0 (876e6e8).
  – New scaling method that combines geometric and equilibrium scaling. Use new value 6 for option int:scaler to activate this.
  – See also the release paper.

3.19.1.4.15 Xpress

• New libraries 32.01.05.

3.19.1.5 Tools

3.19.1.5.1 CDSP

• See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.
3.19.1.5.2 GDXXRW

- Sometimes, Excel is not ready (e.g. because some data needs to be refreshed, when a worksheet is opened), when a read request is sent by GDXXRW. This could cause an exception. With this release we changed the behavior of GDXXRW to wait a second and resend the request in this case. This basically mimics the behavior of setting RWait to 1000 in case of the mentioned exception.

3.19.1.6 Object Oriented APIs

3.19.1.6.1 C++

- Changed the compiler from GCC to Clang on Mac OS X.
- API binaries have been moved to apifiles/C++/lib. On Windows, there are further subdirectories for different compiler versions.
- Update of Visual Studio solutions for the examples on Windows. Three different solutions reflecting the supported versions of Microsoft Visual Studio are available (e.g. examples-vs2013.sln, examples-vs2015.sln, examples-vs2017.sln).
- The GAMS C++ API tutorial has been reworked. Information about building the C++ API examples via cmake, qmake, and Microsoft Visual Studio has been added.
- New example TransportGDX that shows how to import and export GDX files. A description has been added to the tutorial.

3.19.1.6.2 .NET

- Changes for GAMSSymbol and GAMSSymbolRecord: Both classes got the IEqualable<> Interface. As a result the behavior of the Equals function as well as the == and != operators were modified. For both classes Equals and == returns now true, if the internal data reference is the same. Here is an example:

```
GAMSVariable x1 = db.GetVariable("x");
GAMSVariable x2 = db.GetVariable("x");
if (x1 == x2)
    Console.WriteLine("x1 == x2");
else
    Console.WriteLine("x1 != x2");

if (x1.Equals(x2))
    Console.WriteLine("x1 equals x2");
else
    Console.WriteLine("x1 does not equal x2");
```

In previous versions we got this output:

```
x1 != x2
x1 does not equal x2
```

Now we get:

```
x1 == x2
x1 equals x2
```
- New example TransportGDX that shows how to import and export GDX files.
3.19.1.6.3 Java

- The minimum version requirement of the Java Runtime Environment for using the GAMS Java API is now Java SE 7.
- New `TransportGDX` example to demonstrate how to import and export GDX files.
- Removed method `GAMSSymbol.compact`, deprecated since 24.8.1 (December 2016).
- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two symbol objects with the same internal reference are now equivalent, similar to symbol record objects:
  - Two symbols are equivalent if and only if they have the same internal reference.
  - Two symbol records are equivalent if and only if they have the same internal reference.

The behavior of operator `==` remains unchanged. The following example illustrates the new equivalence behavior:

```java
GAMSVariable x1 = db.getVariable("x");
GAMSVariable x2 = db.getVariable("x");
GAMSVariable x3 = x1;
assertTrue(x1.equals(x2)); // true, previously false
assertFalse(x1 == x2); // false, previously false
assertTrue(x1.equals(x3)); // true, previously true
assertTrue(x1 == x3); // true, previously true
```

3.19.1.6.4 Python

- Added implementation of `__eq__()` and `__ne__()` to classes `GamsSymbol` and `GamsSymbolRecord` and its derived classes. As a result the behavior of the operators `==` and `!=` has changed. `==` now returns `True`, if the internal data reference is the same. The behavior of `is` remains unchanged. The following example illustrates the change:

```python
x1 = db["x"]
x2 = db["x"]
print(x1 == x2) # now: True, before: False
print(x1 is x2) # now: False, before: False
```

- New example `transport_gdx.py` that shows how to import and export GDX files. A description has been added to the tutorial.

3.19.1.7 Expert Level APIs

- As announced, the expert-level C++ API files were removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in the last major release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.

  The expert-level Java API files now ensure the load of jni libraries in the similar order as those employed by the Object Oriented Java API: first load from `java.library.path`, if neither specified nor found then from the directory where the API classes are located.

3.19.1.7.1 GDX

- Do not allow empty filename as argument for `gdxOpenAppend`, `gdxOpenRead`, `gdxOpenWrite`, and `gdxOpenWriteEx`. This will create an error right away now.
3.19.1.8 Model Libraries

3.19.1.8.1 GAMS Model Library

- embmiex1.gms: Simple Embedded Code ModelInstance example (417)
- spbenders1.gms: Stochastic Benders - Sequential GAMS Loop (418)
- spbenders2.gms: Stochastic Benders - Async Subsolve GAMS Loop (419)
- spbenders3.gms: Stochastic Benders - Sequential GamsModelInstance (420)
- spbenders4.gms: Stochastic Benders - Parallel MPI (421)
- spbenders5.gms: Stochastic Benders - Parallel MPI with GAMSModelInstance (422)

- Most models in the GAMS Model library have been refurbished to reflect a common syntax style and features introduced since the model was added.
- prime: Make use of the break statement for a little nicer and faster formulation.

3.19.1.8.2 GAMS Test Library

- scensol7.gms: Test GUSS Option ReportLastScen (737)
- emp27.gms: Test LOGMIP/EMP on x.fx=0 handling (738)
- empppy03.gms: Test projection operator when loading data from embedded code (739)
- qcp11.gms: Test dual solution of SOCP (740)
- put11.gms: Put Utility without file handle (741)

3.19.1.9 Solver/Platform availability

<table>
<thead>
<tr>
<th>Solver/Platform</th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COUENNE 0.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 12.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GLOMIQO 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.19.2 25.0.2 Maintenance release (January 31, 2018)

#### 3.19.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Tom Rutherford.

#### 3.19.2.2 GAMS System

##### 3.19.2.2.1 GAMS

- Bugfix: `$/offEmbeddedCode` does not ignore `$/onUNDF` anymore.
- Bugfix: Special values (NA, EPS, INF, ...) are correctly communicated from embedded code back to GAMS.
- Bugfix: Took care about a problem, which caused an unexpected execution error for certain uses of `Put_Utility`.
- Fixed a bug in GMSPython on Mac OS X which prevented the Python interpreter from working. This problem occurred with the DMG installer only.

<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUROBI 7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 10.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LGO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 11.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LOCALLSOLVER 7.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSNLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OQNLP</td>
<td>✓</td>
<td>32bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOLVEENGINE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SOPLEX 3.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS 32.01</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>29.01</td>
</tr>
</tbody>
</table>
3.19.2.3 Solvers

3.19.2.3.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- On Linux, removed the MKL libraries that were optimized to certain x86 instruction sets (AVX, etc.) as this resulted in errors when running ANTIGONE or SCIP, probably due to library conflicts.

3.19.2.3.2 CPLEX

- Option Tuning can be repeated in a GAMS/Cplex option file to provide a larger number of model instances for tuning. Before this change the number was restricted by the maximum line length of an option line (256 characters).

3.19.2.3.3 LocalSolver

- New libraries 7.5 (20180119) for Mac OS X and Windows.

3.19.2.4 Tools

3.19.2.4.1 CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

3.19.2.4.2 CSV2GDX

- Bugfix: Reading sets (no Value or Values option) with the option AutoRow works again.

3.19.3 25.0.3 Minor release (March 21, 2018)

3.19.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Dominik Bongartz, Wolfgang Britz, Erwin Kalvelagen, Maria Kannavou, Hans Mittelmann, Christoph Pahmeyer, Gilles Scouvart, and Hermann Westerholt.

3.19.3.2 GAMS System

3.19.3.2.1 GAMS

- Bugfix for Put Utility: If a put_utility statement was used as the first statement to mark a file as active (so no statement like put fx; was used before) and following put or put_utility statements did not mention this file explicitly, then the file association got lost.

- Bugfix: Avoid a crash when one of the following attributes was used on an empty Singleton Set: .len, .uel, .val (an execution error is generated instead now)

- Bugfix for command line option fileStemApFromEnv: In the past, if the input file was defined including a file extension, the value of this option was extended to the extension and not to the file stem as intended. This is fixed now.

- Bugfix for $if exist File on Windows: In the past, if a file was specified as <Drive>:/<File>, e.g., C:/t.txt (note the forward slash '/') after the colon(':'), this always returned false, even if the file exists. That has been fixed now.
3.19.3.3 Solvers

3.19.3.3.1 BARON

- Fixed use of .EquClass option.

3.19.3.3.2 BONMIN(H), CBC, Couenne

- Branch-and-bound now checks wallclock-time, if running with multiple threads.

3.19.3.3.3 CONOPT4

- New libraries 4.05.
  - Improved the selection of post-triangular variables when there are multiple candidates.

3.19.3.3.4 CPLEX

- Bugfix: If the Cplex conflict refiner (triggered by option iis) could not identify a conflict the behavior was erratic. This has been fixed.
- Bugfix: Value 6 for option MIPStart was documented but got rejected. This has been fixed.

3.19.3.3.5 DICOPT

- Fixed setup of NLP projection problem in feasibility pump.
- Fixed stopping criterion when using stop on crossover: DICOPT was stopping as soon as the relative gap between the objective value of the best known solution and the bound provided by the MIP relaxation was below 0.001, thus might have declared suboptimal solutions as optimal. With this release, DICOPT will use the value of GAMS option optcr as optimality tolerance. Note, that the default for this option is 0.1 (!).

3.19.3.3.6 Ipopt(H)

- New libraries.

3.19.3.3.7 MOSEK

- No update, but be aware that Mosk announced to drop the convex nonlinear optimizer in their next major release (Mosek 9). Thus, in a future GAMS release, GAMS/Mosek will not accept models with model types NLP or DNLP anymore. Note, that linear or quadratic problems (LP, MIP, (MI)QCP) are not affected.
3.19.3.3.8 SCIP

- New libraries 5.0.1 (227c4c7).
  - New option `presolving/symmetry/displaynorbitvars`.

3.19.3.3.9 SoPlex

- New libraries 3.1.1 (ab921a5).

3.19.3.4 Tools

3.19.3.4.1 MODEL2TEX

- Added support for singleton sets.
- Fixed a dependency bug on Mac OS X which prevented `model2tex` from working.
- Fixed a bug regarding SOS variables.
- Fixed a bug regarding domain jumps.
- Added support for Python 3 on Linux and Mac OS X.

3.20 24.9 Distribution

3.20.1 24.9.1 Major release (August 30, 2017)

3.20.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jan Abrell, Etienne Ayotte-Sauvé, Wolfgang Britz, Florian Habermacher, Florian Häberlein, Maximilian Held, Ignacio Herrero, Hanspeter Höschle, Erwin Kalvelagen, Toni Lastusilta, John Ross, Tom Rutherford, and Linus Schrage.

3.20.1.2 Platforms

- The set of supported platforms has not changed, but we've divided it into the core platforms (Windows 32-bit, Windows 64-bit, Linux, and Mac OS X) and the peripheral platforms (AIX, x86-64 Solaris, and Sparc64 Solaris). See Supported Platforms for more details. Please note, however, that we are not changing the content or behavior of GAMS based on this division: GAMS models will continue to work in the same cross-platform way for both core and peripheral platforms.
- We will drop support for x86-64 Solaris with GAMS 25.0.
- We may increase the minimal required GLIBC version on Linux to 2.12 with GAMS 25.0.
- We may increase the minimal required MacOS X version to 10.11 with GAMS 25.0.
3.20.1.3 GAMS System

3.20.1.3.1 GAMS

- New feature, the Embedded Code Facility: This extends the connectivity of GAMS to other programming languages. It allows the use of Python code during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary. The embedded code feature is available on Linux, MacOS X, and Windows. For these platforms, a Python 3.6 installation is included with the GAMS distribution. If the user wants to work with a different Python 3.6, installed separately, for models with embedded code the new command line option `pySetup` needs to be set to 0.

  **Note**
  This feature is currently in beta status. Any feedback to support@gams.com is appreciated.

- New command line option `procDirPath`: Specifies the directory where the process directory should be created.

- New compile time constants to set the model attribute `solPrint`:
  - 0 = `%solPrint.Off%
  - 1 = `%solPrint.On%
  - 2 = `%solPrint.Silent%

- New compile time constants to set the model attribute `solveOpt`:
  - 0 = `%solveOpt.Replace%
  - 1 = `%solveOpt.Merge%
  - 2 = `%solveOpt.Clear%

- Allow new string synonyms to set the integer values of the following command line options:
  - `solPrint`: Off (=0); On (=1); Silent (=2)
  - `sysOut`: Off (=0); On (=1);

- Fixed a bug where the following dollar control options did not get written to the dump file when the command line parameter `dumpOpt` was used. Sometimes this caused an error in the generated file.
  - `$[on|off]DotScale`
  - `$[on|off]Embedded`
  - `$[on|off]Expand`
  - `$[on|off]Local`
  - `$[on|off]Macro`
  - `$[on|off]Margin`
  - `$[on|off]Order`
  - `$[on|off]StrictSingleton`
  - `$[on|off]UNDF`

- Fixed an error which caused the loss of equation information when using the Grid facility in certain cases. This could have caused problems when a hot start should be performed.

- A parameter error is now created if the key of a 'double dash' GAMS parameter exceeds 63 characters (in the past, the key was truncated silently in this case).

- Fixed a crash which happened if the .tl suffix was used on an empty Singleton Set in a put statement.

- Fixed a bug which could have caused a crash if nested loops where used in a particular combination.

- Fixed a bug with `$onPut ... $offPut` in a for and repeat structure: When `$onPut` was the first statement inside one of these programming flow control features, it was executed just once and not repeatedly.
3.20.1.3.2 Special Functions

- Several of the more exotic GAMS intrinsic functions (aka "special functions") were re-implemented to address some bug reports and to pass a more rigorous set of tests. The updated functions include the loggamma, gamma, logbeta, beta, and binomial functions. In general, the updated functions offer improved precision and a more consistent behavior in exceptional cases (e.g. overflows, singularities, and domain violations) compared to the previous versions. In addition, the gamma and beta functions are now classified as smooth (NLP) instead of nonsmooth (DNLP) and the domains of the beta and binomial functions have been changed.

3.20.1.3.3 Documentation

- The contents of GAMS User's Guide and the McCarl (Expanded) User's Guide have been merged, revised, and reorganized as User's Guide as well as A GAMS Tutorial by Richard E. Rosenthal. Also other parts of the documentation has been reorganized and are now more closely integrated.
- The McCarl GAMS User Guide (CHM and PDF) can now be found in the mccarl/ subdirectory in the distribution.
- The PDFs containing the solver manuals and the GAMS User's Guide have been replaced by a single PDF containing large parts of the current documentation. However, the main format for the documentation is HTML.

3.20.1.4 Solvers

3.20.1.4.1 BARON

- New libraries 17.8.7.
  - New range reduction techniques and relaxations for quadratic constraints.
  - New heuristics for finding feasible solutions of integer programs.
  - Improvements in interfaces to local NLP solvers, including the use of second-order derivatives.
  - FilterSQP added to the list of local solvers. New option AllowFilterSQP and added new possible value 14 for option NLPSol.
  - Bugfixes.

3.20.1.4.2 CBC

- New libraries.
  - Fixed some problems with the handling of SOS type 2 in presolve.
- Added option dumpsolutionsmerged to write all found alternate solutions into a single GDX file.

3.20.1.4.3 Conopt

- New libraries 3.17E and 4.03.
  - Fixed a serious error in Conopt4 for exactly threads=8. Moreover, improved multi-threading performance.
  - Major revisions to Conopt4 for reliability and performance.
- The option Rtzern is now user settable for Conopt3.
3.20.1.4.4 CONVERT

- The GAMS equation and variable scale values (suffix .scale) will be communicated to CONVERT independent of the ScaleOpt model attribute.

3.20.1.4.5 CPLEX

- Added option workerAlgorithm to select the method for optimizing Benders subproblems.
- Added option writeAnnotation to create Cplex annotation file.
- The header of MIP/solve trace files contains now the option number and the name of the model.

3.20.1.4.6 DECIS

- DecisC, DecisM, and the EMP-SP solver Decis are now available for MacOS X.

3.20.1.4.7 GAMSCHK

- The procedures that generate output in the listing file are summarized at the end of the GAMSCHK execution. If you use the IDE, these summary lines are clickable and locate the cursor to the corresponding subsection in the IDE. Moreover, the procedures are also entered in the listing file index after the Solution Report.
- The lower bound for options LevelFilt and MargFilt has been reset from 1 to -5.

3.20.1.4.8 GUROBI

- New libraries 7.5.1.
  - Fewer constraint violations in MIQP solutions: Gurobi has tightened the internal tolerances for MIQP models to reduce the number of cases where the solution exhibits small constraint violations.
- New parameter startNodeLimit provides additional control over how much is performed to complete a partial MIP start.
- Gurobi 7.5 will be the last Gurobi release that supports 32-bit Windows. You should plan to migrate your applications to 64-bit Windows in the future.
- The header of MIP/solve trace files contain now the active option number and the name of the model.

3.20.1.4.9 GUSS/Scenario Solver

- Fixed a problem when initializing variables bounds to 0 (updateType=0) that have scenario update parameters for lower and upper bounds.
3.20.1.4.10 Lindo/LindoGlobal

- New libraries 11.0
  - LP Solver Improvements:
    * With new enhancements made to the simplex solvers, the average performance on large instances has increased by 20% for the primal simplex and 15% for the dual simplex compared to the previous version.
  - MIP Solver Improvements:
    * New symmetry detection capabilities to reduce overall branch-bound effort. This may dramatically reduce the time needed to prove optimality on some models with integer variables.
    * Perspective and soft-clique cuts effective on difficult MIQP and models with assignment constraints.
  - Global Solver Improvements:
    * Improved stability and robustness through several enhancements to quadratic recognition and range reduction.
  - Nonlinear Solver Improvements:
    * New major release of the nonlinear solver.
    * Improved preprocessor.
    * Use of interval function and derivative computations.
    * Advanced scaling leading to improved solution quality.

3.20.1.4.11 LocalSolver

- New libraries 7.0 (20170728).
  - Reinforcement of mixed-integer linear programming (LP & MIP) techniques for combinatorial optimization.
  - Reinforcement of nonlinear programming techniques (NLP) for numerical optimization.

3.20.1.4.12 MINOS/QUADMINOS

- New libraries 5.6 (dated July 2016).
  - Improved handling of singularities and empty cols in pivoting code.
  - Further bug fixes and improvements.
- Fix improper handling of scaled CNS models.
- Fix handling of logging frequency.
- QUADMINOS, the quadruple-precision version of MINOS, has been available in previous releases (since 24.4). This release includes the library model [DQQ], an example of how to use MINOS and QUADMINOS together to compute greatly improved solutions at moderate cost.

3.20.1.4.13 MOSEK

- New libraries 8.1.0.23.
  - Performance of the presolve has been improved slightly.
  - Multi-thread performance of the conic optimizer has been improved for certain large models.
  - Changed scaling for quadratic and quadratically constraint optimization problems.
  - Bugfixes.
3.20.1.4.14 MPSGE

- The MPSGE $sysInclude mpsgeset allows now for an optional argument -mt=0 or 1 after the model name. The default value for the argument mt can be controlled via the double dash option --MPSGEMT=0 or 1. If the mt option is set to 1 the MODEL.GEN file is created in the GAMS scratch directory. Hence the $include before the solve needs to read $include "%gams.scrdir%MODEL.GEN". This allows to run multiple MPSGE jobs with the same model in the same working directory. The default of this option is 0. The model hansmge demonstrates the use.

3.20.1.4.15 SBB

- The header of MIP/solve trace files contains now the option number and the name of the model.

3.20.1.4.16 SCIP

- New libraries 4.0 (9d3c1b1).
  - Improved conflict analysis through central conflict pool and dual ray analysis for primal infeasible LPs.
  - New solution polishing to improve integrality of LP solutions when using SoPlex as LP solver.
  - Added adaptive solving behavior of SCIP based on solving phases and heuristic transitions.
  - Revised pseudo random number generation and introduced central random seed for all plugins.
  - Randomized tie-breaking in different parts of the code to reduce performance variability.
  - New primal heuristics GINS, LP face, Complete Sol, Locks, Repair, and Multistart.
  - The 1-opt heuristic is now iterated as long as new incumbents are found.
  - Improved tuning of heuristic timings.
  - Reduced memory usage of primal heuristics that use problem copies.
  - New presolving steps that disaggregate SOC constraints, reformulate QP's by adding KKT conditions, and treat variables appearing only in a single quadratic constraint with proper square coefficients.
  - New separators for gauge cuts, convex projection cuts, and perspective cuts for indicator constraints.
  - Improved knapsack approximation algorithms, greedy knapsack solution for the flow cover separation, clique partitioning, and clique separation.
  - New propagator for OBBDT on convex NLP relaxation.
  - Tuned propagation methods of several constraint handlers and propagation timings.
  - Improved and extended stuffing for linear constraints.
  - Changed handling of coupling constraints for indicator constraints.
  - See also the full release notes, the changelog, and the release paper.
  - Changed parameters:
    * presolving/components/* moved to constraints/components/*
    * conflict/deptscorefac renamed to conflict/graph/deptscorefac
    * misc/permutationseed renamed to randomization/permutationseed
    * misc/permuteconss renamed to randomization/permuteconss
    * misc/permutevars renamed to randomization/permutevars
    * branching/random/seed: default changed from 0 to 41
    * constraints/indicator/sepcouplingcuts: default changed from 0 to 1
    * constraints/SOS1/perfimplanalysis: default changed from 1 to 0
    * heuristics/ofins/freq: default changed from -1 to 0
212 Release Notes

* heuristics/reoptsls/freq: default changed from -1 to 0
* heuristics/trivialnegation/freq: default changed from -1 to 0
* heuristics/clique/initseed: default changed from 0 to 61
* lp/solver: default changed from soplex to soplex2, if CPLEX is not licensed
* presolving/abortfac: default changed from 0.001 to 0.0008
* separating/clique/cliquedensity: default changed from 0.05 to 0
* conflict/usesb: default changed from 0 to 1

- Added option gams/dumpsolutionsmerged to write all found alternate solutions into a single GDX file.
- Changed default for timing/clocktype to wallclock time.
- Initial variable levels can now be passed as partial solution to SCIP. To control the various possibilities, the type of option gams/mipstart has changed from bool to integer. See also subsection Starting point in the GAMS/SCIP solver manual.

3.20.1.4.17 SolveEngine

- New solver SolveEngine to solve LP and MIP problems remotely via the Satalia SolveEngine. The SolveEngine aggregates different solution algorithms for optimization problems and automatically selects an algorithm that seems to suite best for a given model instance.
- GAMS/SolveEngine comes free of charge with any licensed GAMS system. Users must have an API key for the Satalia SolveEngine to submit jobs.

3.20.1.4.18 SoPlex

- New libraries 3.0 (c32c55a).
  - Added a new scaling implementation Least squares (Curtis-Reid scaling).
  - Added persistent scaling to keep scaled LP for multiple reoptimizations.
  - Added an experimental version of a decomposition based approach to avoid degeneracy in the dual simplex method. This feature is activated by setting the parameter bool:decompositiondualsimplex to true, which sets the basis representation to 'row' and the algorithm to 'dual'.
  - New parameter bool:computedegen to enable computation of the degeneracy of the basis in each iteration.
  - New parameter int:printcondition to enable printing the condition number of the basis during solve.
  - Automatically use the row representation for problems with more than 20% more constraints than variables.
  - Changed default for parameter int:factor_update_max from 200 to new value 0.
  - See also the full release notes and the release paper.
- Changed default type of timer to wallclock time.
3.20.1.4.19 XPRESS

- New libraries: Optimizer 31.01.09 (aka XPRESS 8.3). There are many improvements and additions:
  - The parallel MIP code has been completely rewritten to improve performance and scalability.
    * Reduced overhead for small, easy MIPs.
    * Reduced the memory usage for very large MIPs, especially those that are significantly reduced during the initial preprocessing.
    * Heuristics can now be run in parallel with cutting in deterministic mode. Previously, heuristics would only be run in parallel after cutting.
    * Improved implementation of zero-half cutting.
    * Aggregated Mixed Integer Rounding cuts have been improved for network-type problems.
  - New presolve reductions, strengthenings and reformulations for convex quadratic problems.
  - Improved performance and numerical stability of crossover.
  - Code support for AVX2 in the barrier solver: use option cpuPlatform to select the target instruction set.
  - Crossover after a barrier solve is now multi-threaded: see option crossoverThreads for details.
- The header of MIP/solve trace files contains now the option number and the name of the model.

3.20.1.5 Tools

3.20.1.5.1 CSV2GDX

- New option ValueDim: Indicate if an extra dimension for values is added even if there is just one value column. This is ignored, if there is no value column.

3.20.1.5.2 GDXDUMP

- Fixed a problem with writing a scalar variable or equation in CSV format with all fields.

3.20.1.5.3 GDXMERGE

- Improved feedback about problems when processing input files.

3.20.1.6 Object Oriented APIs

- Fixed a bug regarding SymbolUpdateType.Zero that prevented records from being updated in GAMSModelInstance.Solve().

3.20.1.6.1 C++

- This release contains a beta version of the object-oriented C++ API that can be used to control GAMS from within C++11 and later. It allows the seamless integration of GAMS into C++ applications by providing appropriate classes for the interaction with GAMS. The GAMSDatabase class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the GAMSJob class and by using the GAMSModelInstance class a sequence of closely related model instances can be solved in an efficient way.
- The API is available in the distributions for Linux, MacOS X, and Windows.
- Furthermore, the C++ API is published under MIT license and is hosted at the GAMS GitHub organization.
- To use this API please check the GAMS API documentation.
3.20.1.6.2 .NET

- New functions `GAMSWorkspace.AddJobFromApiLib`, `GAMSWorkspace.AddJobFromNoaLib` to create `GAMSJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library.

3.20.1.6.3 Python

- New functions `GamsWorkspace.add_job_from_apilib`, `GamsWorkspace.add_job_from_noalib` to create `GamsJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library.
- Added a version check for the setup.py scripts to avoid unintentional installation of wrong versions.

3.20.1.7 Expert Level APIs

- With GAMS 25.0, the expert-level C++ API files will be removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in this release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.

3.20.1.7.1 GAMS Modeling Object

- Changed the function `gmoGetModelTypeTxt`: Added argument for model type number instead of using the model type of the stored model.

3.20.1.8 Model Libraries

3.20.1.8.1 GAMS Data Library

- `embeddedSort.gms`: Sorting of numerical data using the embedded code facility (108)
- `embeddedSplit.gms`: Splitting of labels using the embedded code facility (109)
- `embeddedMultiInstance.gms`: Handling multiple instances of the embedded code facility at once (110)

3.20.1.8.2 GAMS Model Library

- `dqq.gms`: Warm-starting quad-precision MINOS (414)
- `cbenders.gms`: Cplex Benders for a Simple Facility Location Problem (415)
- `robustlp.gms`: Robust linear programming as an SOCP (416)
3.20.1.8.3 GAMS Test Library

- mpsge14.gms: MPSGE sync test: .GEN/integer1/nnsoves (713)
- fn_sincosinvr.gms: Test sin/cos func/grad interval evals (714)
- procdir1.gms: Test correct behavior of procDir, scrDir and procDirPath (715)
- empbp06.gms: Bilevel model with phantom vars owned by leader (716)
- fnsl.exp.gms: Test correctness of slexp intrinsic (717)
- fn.slexp2.gms: Test correctness of slexp intrinsic (718)
- fn.sqexp.gms: Test correctness of sqexp intrinsic (719)
- fn.sqexp2.gms: Test correctness of sqexp intrinsic (720)
- fn.slog10.gms: Test correctness of slog10 intrinsic (721)
- fn.slog102.gms: Test correctness of slog10 intrinsic (722)
- fn.sqlog10.gms: Test correctness of sqlog10 intrinsic (723)
- fn.sqlog102.gms: Test correctness of sqlog10 intrinsic (724)
- fn.srec.gms: Test correctness of srec intrinsic (725)
- fn.srec2.gms: Test correctness of srec intrinsic (726)
- fn.sqrec.gms: Test correctness of sqrec intrinsic (727)
- fn.sqrec2.gms: Test correctness of sqrec intrinsic (728)
- fn.loggamma.gms: Test correctness of loggamma intrinsic (729)
- fn.binomial.gms: Test correctness of binomial intrinsic (730)
- breakcont2.gms: Advanced test for break and continue statements (731)
- fn.logbeta.gms: Test correctness of logbeta intrinsic (732)
- fn.beta.gms: Test correctness of beta intrinsic (733)
- ssuffix.gms: List of all System Suffixes (734)
- embpy01.gms: Test for embedded code facility (735)
- embpy02.gms: Test for embedded code facility (736)

3.20.1.9 Solver/Platform availability

<table>
<thead>
<tr>
<th></th>
<th>x86 32bit MS Windows</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
<th>x86 64bit SOLARIS</th>
<th>Sparc 64bit SOLARIS</th>
<th>IBM Power 64bit AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAECP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BDMLP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BONMIN 1.8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 3.20.2 24.9.2 Minor release (November 14, 2017)

#### 3.20.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Paul Buckland, William N. Caballero, Xin Fang, Christophe Gouel, Fabricio Porras-Ortiz, Shigeru Tsubakitani, and Haoshui Yu.
3.20.2.2 GAMS System

3.20.2.2.1 GAMS

- Added prefixing of PYTHONPATH with the site-packages directory of the GAMS Python 3.6 installation in GMSPython.
- Added support for Python sets in the Embedded Python Code method gams.set().
- Speed up detection/extraction of quadratic terms for QCP models.
- Do not allow the use of Embedded Code if execMode is set to 2 or higher.

3.20.2.2.2 Installer

- Removed the execution of a Python script during the Windows installation process that patches the path of certain Python programs in GMSPython\Scripts.

3.20.2.3 Solvers

3.20.2.3.1 BARON

- New libraries 17.10.16.
- Due to a problem with the Ipopt interface in BARON on Mac OS X, the use of Ipopt in BARON is currently disabled on Mac OS X.
- GAMS can now pass a term like \((-\text{negativeConstant})^\text{negativeIntegerVariable}\) to BARON.

Note

By default, GAMS correctly rejects such a formulation, though. To work around this, MaxExecError needs to be set to a positive number and the option sys12 needs to be set to 1. Here is an example, how this could be used:

```plaintext
scalar a /-2/;
integer variable x;
x.lo = -3;
x.up = 3;

variable z;
equation e;
e.. z =e=a**x;
model m /e/;
x.l=1;
MaxExecError = 10;
option sys12 = 1;
solve m min z use minlp;
```

3.20.2.3.2 CPLEX

- Removed any limits on the number of Threads in a GAMS/CPLEX option file. Previously, the maximum for option Threads was 128.
3.20.2.3.3 CONOPT

- New libraries 3.17F.
  - Fixed an infinite loop when using option LMMXSF.

3.20.2.3.4 CONVERT

- Fix for variables with negative lower and infinite upper bound in LINGO output. The LINGO output has been changed to always use the @Bnd keyword to write variable bounds.

3.20.2.3.5 GUROBI

- New libraries 7.5.2.

3.20.2.3.6 GUSS/Scenario Solver

- Added the GUSS option ReportLastScen. If this is set to 1 the solution of the last scenario will be reported back to GAMS rather than solution of the base case. This is particularly useful when GUSS is used to implement a homotopy approach where the shock to a parameter is sliced in several small shocks and hence the last scenario represents the solution to the shocked system.

3.20.2.3.7 Lindo/LindoGlobal

- New libraries 11.0.300.

3.20.2.3.8 LogMIP

- Fixed improper handling of variables fixed at zero that occurred if they were used in nonlinear equations in a disjunction reformulated via the convex hull.

3.20.2.3.9 Mosek

- New libraries 8.1.0.31.
- Fixed dual solution for conic quadratic problems.

3.20.2.3.10 SCIP

- New libraries 4.0 (#22b4564).

3.20.2.3.11 SoPlex

- New libraries 3.0 (#3bfa247).
3.20.2.4 Object Oriented APIs

3.20.2.4.1 C++

- Added a try/catch block around the code that uses the classes from the GAMS C++ OO-API in all examples. Without this, the exceptions thrown by the methods of the GAMS classes will not be reported and harmless exceptions, like a compilation or execution error in the GAMSJob.run method result in application crashes.

3.20.2.4.2 Python

- Fixed a bug in the setup script that prevented files from being installed in the correct location in certain cases.

3.21 24.8 Distribution

3.21.1 24.8.1 Major release (December 21, 2016)

3.21.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Etienne Ayotte-Sauvé, Wolfgang Britz, Göran Bylund, Wietse Dol, Gregory Dourbois, Katja Jensen, Josef Kallrath, Toni Lastusilta, Renger van Nieuwkoop, and Tom Rutherford.

3.21.1.2 Platforms

- On Linux, the minimal required GLIBC version is now 2.7.
- On MacOS X, for some solvers (ANTIGONE, BARON, CPLEX, COIN-OR, SOPLEX, SCIP) the minimal required Mac OS X version is now 10.10. We will drop support for older Mac OS X versions for the complete GAMS system with the next major release.
- A generic license (with platform code GEN) is now limited to the platforms Windows, Linux, and Mac OS X. If you have a generic license and need a license for one of the other platforms (e.g. AIX), please contact sales or support to receive an additional license file for that platform free of charge.

3.21.1.3 GAMS System

3.21.1.3.1 GAMS

- The statements Break and Continue were added to allow more control over the execution of control structures (loop/while/repeat/for):
  - Break [n];: Terminate the n inner most control structures (n is optional, if it is omitted, it is set to 1).
  - Continue;: Jump to the end of the inner most control structure.
• Extend the report summary of the solution report to print the number of variables and equations where the level was projected to one of the bounds (compare model attribute tolProj), if that number is greater than 0.

• The model attribute tolProj to project levels to bounds is no longer ignored when the GAMS Grid Facility (SolveLink = 3 or 4) is used.

• New command line option fileStem: Sets the file stem for output files which use the input file name as stem by default, see fileStem for more details.

• New option MCPRHoldfx (can be set using the option statement, a command line parameter, or model attribute):
  – Prints list of rows that are perpendicular to variables removed due to the holdfixed setting if set to 1.

• New option AsyncSollst (can be set using the option statement, or a command line parameter):
  – If set to 1, GAMS prints a solution listing to the lst file also when an asynchronous solve (see The Grid and Multi-Threading Solve Facility) is used.
  – The default value is 0.

• New variant for the SolveLink option and model attribute: 7 (compile time constant %solveLink.Threads Simulate%) - The problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6 (see Multi-threading Submission Testing)

• New function numCores (available at both compile and execution time): Returns the number of logical cores in the system.

• Allow macro expansion in the domain list of a symbol declaration.

• New dollar control splitOption that splits a string representing a option/value pair into option name and option value.

• Fixed a problem with the optional maxWait parameter of the function readyCollect.

• Fixed potential error in file created by dumpOpt, if there were symbols loaded from GDX.

• Fixed an error which could lead to wrong results when assigning to a symbol while the symbol’s alias was used on the right hand side of the assignment.

• Fixed an error which could lead to a crash if a phantom set element was used to control an assignment inside a loop.

• Fixed a problem with unscaling solutions for MCP models that resulted in incorrect dual values for scaled MCPs. For an example, compare the new test library model mcp11 when run with 24.8 and with something previous.

3.21.1.3.2 Documentation

• The offline documentation now provides search and keyword indexing functionalities in addition to navigating the documentation.

• The Microsoft Compiled HTML help file gams.chm is no longer available.

• The table of contents for browsing the GAMS model libraries is now available offline in HTML format.
3.21.1.4 Solvers

3.21.1.4.1 BARON


3.21.1.4.2 CPLEX

- New libraries 12.7.0.0.
- Support for the platform MS Windows 32 bit, Solaris i86pc, and Solaris SPARC 64bit has been dropped by IBM for Cplex 12.7. The GAMS system for these platforms contains Cplex 12.6.3.
- Cplex 12.7 implements Benders Algorithm. This is available in CplexD only.
- The IIS option triggers now the conflict refiner. The IIS function in Cplex 12.7 has been replaced by the conflict refiner. The IIS option now also works on infeasible models with model status Infeasible No Solution as well as on problems with discrete variables.
- New parameters
  - BendersFeasCutTol: Tolerance for whether a feasibility cut has been violated in Benders decomposition
  - BendersOptCutTol: Tolerance for optimality cuts in Benders decomposition
  - BendersStrategy: Benders decomposition algorithm as a strategy
  - DataCheck: Data consistency checking and modeling assistance
  - RLTcuts: Reformulation Linearization Technique (RLT) cuts
- Modified parameters:
  - MipStart: A value of 6 accepts the (partial) MipStart without any checks

3.21.1.4.3 CONOPT

- This distribution includes the first official release of the CONOPT4 solver. The chapter in the solver manual has an interesting subsection on when you should use CONOPT4 for all existing CONOPT users.
- The current solver alias CONOPT still points to the CONOPT3 solver. This might change in the next, or next but one major release. We invite you to share your experience with this new version of CONOPT via support (support@gams.com) with us.

3.21.1.4.4 ConvertD

- Write proper scale and prior information for equations and variables with option Jacobian.

3.21.1.4.5 DICOPT

- An implementation of a Feasibility Pump primal heuristic (for convex MINLP) has been added to DICOPT. This heuristic can be run before the actual DICOPT algorithm. Outer approximation cuts from the MIP subproblem of the Feasibility Pump are transferred to initialize the MIP outer approximation of the main DICOPT algorithm. Option feaspump can be used to enable the Feasibility Pump and various other options are available to tune the feasibility pump.
- Option convex has been added to indicate the presence of convex MINLP. If this option is set, the defaults for various other options are modified.
3.21.1.4.6 GUROBI

- New libraries 7.0.1.
- Explore alternative solutions via the Solution Pool.
- New parameters
  - BestObjStop: Objective value to stop optimization
  - BestBdStop: Objective bound to stop optimization
  - InfProofCuts: Infeasibility proof cut generation
  - StrongCGCuts: Strong-CG cut generation
  - DegenMoves: Degenerate simplex moves
  - TuneCriterion: Tuning criterion
  - SolNPool: Activate export of alternative solution
  - PoolSolutions, PoolSearchMode, and PoolGap to control generation of alternative solutions
- Gurobi 7 supports general constraints. With the help of the dot-option .GenConstrType the user can classify a constraint to be of type Max, Min, Abs, And, and Or.
- Gurobi 7 supports indicator constraints.
- Gurobi 7 supports multi objective hierarchical optimization. Details can be found in the GAMS/Gurobi manual, in subsection Multiple Objectives.

3.21.1.4.7 JAMS

- New option ZipDebug=xxx.zip to specify that, in the event of abnormal termination or behavior, a zip file of debugging info be created.
- New reformulation options to handle shared or duplicated equations and variables in equilibrium models.

3.21.1.4.8 Knitro

- New libraries 10.2.0.
  - Significant speed and robustness improvements when using BFGS (hessopt=2) or L-BFGS (hessopt=6) Hessian approximations with the default Knitro interior-point method.
  - General performance improvement on mixed-integer models.
  - Minor bug fixes.
  - New mixed-integer SQP (MISQP) algorithm for nonlinear mixed-integer models (mip_method=3). This new algorithm is intended for small, potentially non-convex models with possibly expensive function evaluations. It can be used even when integer variables are not relaxable (i.e. functions can only be evaluated with integer variables at integer points) by setting mip_relaxable=0, and can be used with parallel multistart.
  - New option mip_nodealg to control algorithm used at B&B nodes: overrides the generic algorithm option.
- The Knitro Tuner is now available from the GAMS/Knitro link.
3.21.1.4.9 Lindo/LindoGlobal

- New libraries 10.0.131.
  - LP Solver Improvements:
    * With new enhancements made to the simplex solvers, the average performance on large instances has increased by 35% for the primal simplex and by 20% for the dual simplex compared to the previous version.
  - MIP Solver Improvements:
    * New heuristic algorithms help to find significantly better solutions for many models with knapsack constraints and block structures.
    * New MIP preprocessing level devoted to tightening variable bounds for some nonlinear models.
  - Stochastic Solver Improvements:
    * Improved cut management for Nested Benders Decomposition Method leading to speed improvements over 60% for large linear multistage SP instances.
    * Better handling of multistage SP models which do not have full-recourse.
  - Global Solver Improvements:
    * Incorporates bound tightening process to the linearization procedure and improve solvability of linearized model.

3.21.1.4.10 LocalSolver

- New libraries 6.5 (20160729).
  - For near-linear (discrete or continuous) problems, new moves have been introduced based on linear programming and mixed-integer linear programming techniques. These moves allow to intensify the search on near-linear models by exploring optimally larger neighborhoods.
- Added possibility for "Hierarchical Optimization of Multiple Objective Functions".
- Removed option origlog. GAMS/LocalSolver will now always print the original LocalSolver log.
- Fixed level values of aggregated variables in solution.

3.21.1.4.11 Mosek

- New libraries 8.0.0.48 (Mosek release notes)
  - Presolve performance has been improved.
  - The eliminator in the presolve has been reimplemented, is usually faster, and requires much less memory.
  - Presolve has been improved on conic quadratic problems.
  - The numerical stability of conic optimizer has been improved significantly, particularly for semidefinite optimization problems.
  - The scaling routine for the conic optimizer is more aggressive.
  - Quadratic and quadratically constrained problems are now internally converted to conic form and are solved using the conic optimizer. Nevertheless full primal and dual information to the original problems is available.
  - A dualizer for conic quadratic problems is now available. By default it dualize the problems before optimizing if deemed worthwhile. The dualization is transparent to the user and can be turned off.
The conic optimizer linear algebra is now parallelized using Cilk Plus and scales better when the number of threads is increased for large problems. Moreover, for smallish problems using too many threads does not hurt performance.

The computational efficiency graph partitioning based ordering method in the interior-point optimizer has been improved.

It is now possible to force the interior-point optimizer to run in the calling thread.

Only one mixed integer optimizer is available now, which corresponds to the mixed integer conic optimizer that was introduced with version 7.

The primal network simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.

The primal-dual simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.

The concurrent optimizer has been removed. It is suggested to use the interior-point optimizer instead.

The following GAMS/Mosek options have been removed:

- MSK_IPAR_ANA_SOL_BASIS
- MSK_IPAR_ANA_SOL_PRINT_VIOLATED
- MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS
- MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX
- MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX
- MSK_IPAR_CONCURRENT_PRIORITY_INTPNT
- MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX
- MSK_DPAR_FEASREPAIR_TOL
- MSK_IPAR_FEASREPAIR_OPTIMIZE
- MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL
- MSK_IPAR_INTPNT_FACTOR_METHOD
- MSK_IPAR_LOG_CONCURRENT
- MSK_IPAR_LOG_NONCONVEX
- MSK_IPAR_LOG_PARAM
- MSK_IPAR_LOG_SENSITIVITY
- MSK_IPAR_LOG_SENSITIVITY_OPT
- MSK_IPAR_MAX_NUM_WARNINGS
- MSK_IPAR_MIO_CONT_SOL
- MSK_IPAR_MIO_CUT_CG
- MSK_IPAR_MIO_CUT_LEVEL_ROOT
- MSK_IPAR_MIO_CUT_LEVEL_TREE
- MSK_IPAR_MIO_FEASPUMP_LEVEL
- MSK_DPAR_MIO_HEURISTIC_TIME
- MSK_IPAR_MIO_HOTSTART
- MSK_IPAR_MIO_KEEP_BASIS
- MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER
- MSK_DPAR_MIO_MAX_TIME_APRX_OPT
- MSK_IPAR_MIO_PRESOLVE_AGGREGATE
- MSK_IPAR_MIO_PRESOLVE_PROBING
- MSK_IPAR_MIO_PRESOLVE_USE
- MSK_DPAR_MIO_REL_ADD_CUT_LIMITED
- MSK_IPAR_MIO_STRONG_BRANCH
- MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS
- MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS
- MSK_DPAR_MIO_TOL_REL_RELAX_INT
The following GAMS/Mosek options have been added:

- `MSK_DPAR_DATA_SYM_MAT_TOL`: Absolute zero tolerance for elements in symmetric matrices.
- `MSK_DPAR_DATA_SYM_MAT_TOL_HUGE`: An element in a symmetric matrix which is larger than this value in absolute size causes an error.
- `MSK_DPAR_DATA_SYM_MAT_TOL_LARGE`: An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.
- `MSK_DPAR_INTPNT_QO_TOL_DFEAS`: Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
- `MSK_DPAR_INTPNT_QO_TOL_INFEAS`: Controls when the conic interior-point optimizer declares the model primal or dual infeasible.
- `MSK_DPAR_INTPNT_QO_TOL_MU_RED`: Relative complementarity gap feasibility tolerance used when interior-point optimizer is applied to a quadratic optimization problem.
- `MSK_DPAR_INTPNT_QO_TOL_NEAR_REL`: Termination tolerance multiplier that is used if no accurate solution can be found.
- `MSK_DPAR_INTPNT_QO_TOL_PFEAS`: Primal feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
- `MSK_DPAR_INTPNT_QO_TOL_REL_GAP`: Relative gap termination tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
- `MSK_IPAR_INTPNT_MULTI_THREAD`: Controls whether the interior-point optimizers can employ multiple threads if available.
- `MSK_IPAR_MIO_CUT_CLIQUE`: Controls whether clique cuts should be generated.
- `MSK_IPAR_MIO_CUT_GMI`: Controls whether GMI cuts should be generated.
- `MSK_IPAR_MIO_CUTIMPLIED_BOUND`: Controls whether implied bound cuts should be generated.
- `MSK_IPAR_MIO_CUT_KNAPSACK_COVER`: Controls whether knapsack cover cuts should be generated.
- `MSK_IPAR_MIO_CUT_SELECTION_LEVEL`: Controls how aggressively generated cuts are selected to be included in the relaxation.
- `MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE`: Enables or disables perspective reformulation in presolve.
- `MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL`: Controls whether presolve can be repeated at root node.
- `MSK_IPAR_MIO_VB_DETECTION_LEVEL`: Controls how much effort is put into detecting variable bounds.
- `MSK_IPAR_OPF_WRITE_HEADER`: Write a text header with date and MOSEK version in an OPF file.
* **MSK_IPAR_OPF_WRITE_HINTS**: Write a hint subsection with problem dimensions in the beginning of an OPF file.
* **MSK_IPAR_OPF_WRITE_PROBLEM**: Write objective, constraints, bounds etc.
* **MSK_IPAR_OPF_WRITE_SOL_BAS**: Whether to include basic solution in OPF files.
* **MSK_IPAR_OPF_WRITE_SOL_ITG**: Whether to include integer solution in OPF files.
* **MSK_IPAR_OPF_WRITE_SOL_ITR**: Whether to include interior solution in OPF files.
* **MSK_IPAR_OPF_WRITE_SOLUTIONS**: Enable inclusion of solutions in the OPF files.
* **MSK_DPAR_SEMIDEFINITE_TOL_APPROX**: Tolerance to define a matrix to be positive semidefinite.
* **MSK_IPAR_WRITE_DATA_PARAM**: If this option is turned on the parameter settings are written to the data file as parameters.
* **MSK_IPAR_WRITE_LP_FULL_OBJ**: Write all variables, including the ones with 0-coefficients, in the objective.
* **MSK_IPAR_WRITE_MPS_FORMAT**: Controls in which format the MPS is written.
* **MSK_IPAR_WRITE_TASK_INC_SOL**: Controls whether the solutions are stored in the task file too.
* **MSK_SPAR_WRITE_LP_GEN_VAR_NAME**: Sometimes when an LP file is written additional variables must be inserted.

- For the following options, the default value has changed:
  * **MSK_DPAR_MIO_TOL_FEAS** from 1e-7 to 1e-6
  * **MSK_IPAR_LOG_MIO_FREQ** from 1000 to 10
  * **MSK_IPAR_WRITE_PRECISION** from 8 to 15

- Fixed selection of optimizer for "fixed solve" of a mixed-integer conic problem.

### 3.21.1.4.12 Pyomo

- Fixed compatibility issues with different versions of Pyomo.

### 3.21.1.4.13 SCIP

- New libraries 3.2 (#0d4fc08).
- Changed the default LP solver to SoPlex also for the case where a CPLEX license is available, due to problems when using CPLEX 12.7.0.0 as LP solver in SCIP.

### 3.21.1.4.14 SoPlex

- The GAMS/SoPlex interface has been rewritten and does not use the OsiSpx layer anymore. The solver OSISOPLEX is now an alias for the solver SoPlex.
- SoPlex parameter files can now be used.
- SoPlex can now be warmstarted when only the model instance data changes (e.g., via GUSS).
- New libraries 2.2 (df190de).
3.21.1.5 Tools

3.21.1.5.1 CSV2GDX

- Improved error reporting.

3.21.1.5.2 GDXDUMP

- New command line option `CSVAllFields` to get all fields (level, marginal, lower, upper, and scale) when writing a variable or equation symbol in CSV format.

3.21.1.5.3 MODEL2TEX

- Increased the page width of the txt file generated by the GAMS command line option `docfile` to 32767 (max. value).
- Changed the default encoding to `latin` and added a new command line parameter that allows to change the encoding (`-e=ENCODING`)
- Changed the default format of the symbol tables.
- The JSON style file contains a new property called `columnSetting` that allows to adjust the columns.
- Avoid some unnecessary parentheses in sum and product operators.
- Minor change in the equations subsection that removes several warnings.
- Several minor bug fixes.

3.21.1.5.4 GDXXRW

- Fixed a problem when writing a symbol with option merge or clear to a range with `CDim=0` or `RDim=0`.

3.21.1.5.5 GMSZIP/GMSUNZIP

- New versions of Info-ZIP’s tools `zip` (version 3.1c02) and `unzip` (version 6.00). The executable names have been prefixed with ”gms” for clear identification.

3.21.1.6 Object Oriented APIs

- New option `GAMSOptions.ErrorLog`: Maximal number of error message lines written to the log for each error.

3.21.1.6.1 .NET

- New examples that demonstrate the use of the API in a graphical environment: TransportGUI, CutstockGUI, FarmGUI.
- Distribute compiled GUI examples in directory `<GAMSDir>\apifiles\GUIexamples` on Windows.
- `GAMSWorkspace.AddJobFromFile`: Throw an exception if a given file does not exist.
3.21.1.6.2 Python

- The option `GamsOptions.errorlog` has been renamed to `GamsOptions.errorlog` in order to indicate it as public.

3.21.1.7 Model Libraries

3.21.1.7.1 GAMS API Library

- `PInterrupt.gms`: Test GamsJob interrupt mechanism in object oriented Python API (54)
- `CSInterrupt.gms`: Test GamsJob interrupt mechanism in object oriented .Net API (55)

3.21.1.7.2 GAMS Model Library

- `asyncloop.gms`: Transportation Problem with async loop body execution (411)
- `trnsindic.gms`: Fixed Charge Transportation Problem with Indicator Constraints (412)
- `timesteps.gms`: Accessing previous (or next) time steps in an equation fast (413)
- Modified models:
  - Make use of the new `break` statement to formulate the following models a little nicer: `tsp1`, `tsp4`, `cutstock`, `awktsp`, `sddp`, `sipres`, `allbases`, `qfilter`
  - `tgridmix`: The logic of submission and collection has been reworked so that the scenario sets do not have to be one dimensional and ordered (contributed by Tom Rutherford).
  - `asyncjobs`: The logic for putting out the log line in time intervals was flawed and has been corrected.

3.21.1.7.3 GAMS Test Library

- `emp17.gms`: Simple test of one optimizing agent (693)
- `breakcont1.gms`: Test break and continue statements (694)
- `lindgl04.gms`: Test non-convex quadartic program with Lindo(Global) (695)
- `mcp11.gms`: Test marginals for a scaled MCP problem (696)
- `csv2gdx1.gms`: Test csv2gdx on input containing spaces (697)
- `gdxxrw7.gms`: Test merge and clear option for special data layout (698)
- `scale02.gms`: Test that an MCP with scales is rejected when appropriate (699)
- `emp18.gms`: Test JAMS/EMP on implicit variable handling (700)
- `emp19.gms`: Test JAMS/EMP on implicit variable handling (701)
- `emp20.gms`: Test JAMS/EMP on implicit variable handling (702)
- `emp21.gms`: Test JAMS/EMP on implicit variable handling (703)
- `emp22.gms`: Test JAMS/EMP on implicit variable handling (704)
3.21 24.8 Distribution

- emp23.gms: Test JAMS/EMP on implicit variable handling (705)
- emp24.gms: Test JAMS/EMP on implicit variable handling (706)
- emp25.gms: Test JAMS/EMP on implicit variable handling (707)
- emp26.gms: Test JAMS/EMP on implicit variable handling (708)
- gurobi02.gms: GUROBI test suite - general constraints max,min,abs (709)
- gurobi03.gms: GUROBI test suite - general constraints and,or (710)
- slx01.gms: run tests for different solvelink values (711)
- gurobi04.gms: GUROBI test suite - multi objective (712)

3.21.2 24.8.2 Maintenance release (January 03, 2017)

3.21.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Evangelos Panos.

3.21.2.2 GAMS System

3.21.2.2.1 GAMS

- Change for command line parameter MultiPass: If it is set to 2, all errors from \$gdxIn are ignored now
- Fixed a bug which caused wrong results in some assignments which use symbols that were used out-of-order in previous assignments.

3.21.2.3 Solvers

3.21.2.3.1 SCIP

- Added a workaround that allows for using CPLEX 12.7.0.0 as LP solver in SCIP again. For the moment, the default is still to use SoPlex.

3.21.3 24.8.3 Minor release (January 28, 2017)

3.21.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Mohammad R. Hesamzadeh, and Katja Jensen.

3.21.3.2 GAMS System

3.21.3.2.1 GAMS

- Fixed potentially wrong values for model attribute etSolver for cases where the command line option solveLink was set to 1, 2 or 5.
3.21.3.3 Solvers

3.21.3.3.1 BARON

- New libraries 17.1.2.
  - More robust links with COIN-OR solvers and a better link with FICO Xpress.
  - Some new RLT cuts are included in relaxations.

3.21.3.3.2 DE

- Create a capability error if DE is called to solve a model of type EMP without stochastic information.

3.21.3.3.3 DECIS

- Create a capability error if DECIS is called to solve a model of type EMP without stochastic information.

3.21.3.3.4 Examiner2

- Fix behavior when the subsolver returns a model status like 19 Infeasible - No Solution. The model status can be passed unchanged back to GAMS in such a case.

3.21.3.3.5 Kestrel

- Adjusted the default URL to https using port 3333.
- Fixed a bug that prevented to specify the protocol in the neos_server parameter in an option file. The complete format of the parameter is now protocol://host:port.

3.21.3.3.6 LINDO

- Create a capability error if LINDO is called to solve a model of type EMP without stochastic information.

3.21.3.3.7 MOSEK

- New libraries 8.0.0.53.

3.21.3.3.8 SCIP

- Changed the default LP solver back to CPLEX, if available (see also 24.8.1 and 24.8.2 release notes).
3.21.3.4 Tools

3.21.3.4.1 GDXDUMP

- Fixed wrong output in case of nested quotes in symbol text.

3.21.3.5 Model Libraries

3.21.3.5.1 GAMS Model Library

- linearne: Minor fix to model formulation (contributed by Mohammad R. Hesamzadeh).


3.21.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Ivan Leung, Tara Rengarajan, and Sajjad Shafiei.

3.21.4.2 GAMS System

3.21.4.2.1 GAMS

- Fixed an issue which caused unnecessary memory consumption if many indexed symbols with explicit labels were used.

3.21.4.3 Solvers

3.21.4.3.1 BARON

- New libraries 17.4.1.
  - Bug fixes and an enhanced convexity detector.

3.21.4.3.2 CPLEX

- New libraries 12.7.1.0.
  - Note that with this update the log generated by Cplex changed in a way that the Cplex options set (either by the user or by GAMS defaults) are displayed. This is intentional and no sign of a problem. So one could see something like this in the log when running GAMS/Cplex:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPXPARAM_Simplex_Limits_Iterations</td>
<td>2000000000</td>
</tr>
<tr>
<td>CPXPARAM_TimeLimit</td>
<td>1000</td>
</tr>
<tr>
<td>CPXPARAM_Threads</td>
<td>1</td>
</tr>
</tbody>
</table>
3.21.4.3.3 DICOPT

- Fixed serious bug in feasibility pump implementation.

3.21.4.3.4 Examiner, Examiner2

- The list of checks to perform for solved models was set incorrectly when the trace option was used.

3.21.4.3.5 GUROBI

- New libraries Gurobi 7.0.2.
- Added options MultiObjMethod and MultiObjPre.
- Fixed a problem that lead to wrong error messages when setting the ObjNAbsTol and ObjNRelTol parameters.
- Fixed a problem that reported back the wrong alternative solution.

3.21.4.3.6 JAMS

- Fix problem with bilevel models having variables owned by the leader but not appearing in the leader objective or constraints.

3.21.4.3.7 Lindo/LindoGlobal

- New libraries 10.0.179.

3.21.4.3.8 MOSEK

- New libraries 8.0.0.60.

3.21.4.3.9 NLPEC

- Fix problem handling empty constraints (e.g. \( \text{eps} x = \text{G} = 0 \)) that appear in MPEC models. Note that such models can easily be produced by JAMS/EMP.

3.21.4.4 Object Oriented APIs

3.21.4.4.1 Python

- Added support for Python 3.6.
- Fixed a bug in `GamsJob.run()` that prevented the underlying GAMS model from terminating, if executables spawned by GAMS generate log output that is not captured.
3.21.5 24.8.5 Maintenance release (May 10, 2017)

3.21.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Ana Carolina, Gary Goldstein, Erwin Kalvelagen, Amit Kamdia, Toni Lastusilta, Antti Lehtila, Bruce McCarl, and Hans Kristian Ringkjob.

3.21.5.2 Platforms

- The AIX system is now build on AIX 7.1.

3.21.5.3 GAMS System

3.21.5.3.1 GAMS

- Fixed a bug which could cause a crash in particular assignments involving singleton sets or explicit labels. This was introduced with GAMS 24.8.4.
- Fixed problems with the break statement:
  - There was a potential crash if break was used in a "sparse loop", e.g., \texttt{loop(j$x(j), ...).}
  - If break was used in a loop with more than one index, e.g., \texttt{loop((i,j), ...)}, that loop was treated as multiple loops for the break statement, one for each index. Now it is treated as just one loop as intended.
  - Fixed a problem where a loop was not correctly recognized as a loop if it runs over just one fixed element, e.g., \texttt{loop(i('i2'), ...)}.

3.21.5.4 Solvers

3.21.5.4.1 BARON

- Initialize BARON option Threads with value of GAMS option Threads, if the latter is at least 1.

3.21.5.4.2 CBC

- New libraries.
  - Fixed a bug in presolve that caused problems with fixed discrete variables.

3.21.5.4.3 GUROBI

- Suboptimal solutions were not reported back to GAMS. This has been fixed.

3.21.5.4.4 Lindo/LindoGlobal

- New libraries 10.0.182.
3.21.5.4.5 MOSEK

- New libraries 8.0.0.69.

3.22 24.7 Distribution

3.22.1 24.7.1 Major release (March 14, 2016)

3.22.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Alex Dowling, Johannes Hedtrich, Austin Milt, and Salvador Pineda.

3.22.1.2 GAMS System

3.22.1.2.1 GAMS

- Be more strict when the syntax $\texttt{LOAD par=var.L}$ is used: This creates an error now, if \texttt{par} is not a parameter (if \texttt{par} was a variable before, the complete variable \texttt{var} was loaded as if the suffix was omitted).
- GAMS Grid Facility (\texttt{SolveLink = 3 or 4})
  - Fixed a bug for the model attribute \texttt{objval}: This could have been wrong when the objective variable was scaled.
  - Fixed a bug for the marginals: These could have been wrong when the objective variable was scaled.
- Solving in parallel threads (\texttt{SolveLink = 6})
  - Allow to solve scaled models (model attribute \texttt{scaleOpt} is set to 1) as well (this generated an execution error before).

3.22.1.3 Solvers

3.22.1.3.1 BARON

- New libraries 16.3.4.
  - Faster LP interfaces.
  - Additional checks on solution reliability of LP and NLP subsolvers.
  - New automatic differentiation routines.
  - Improved handling of memory-intensive problems.
  - Enhanced presolving for continuous and integer programs.
  - New primal heuristic for integer programs.
  - New facilities for solving constraint (non)linear systems (CNS).
3.22.1.3.2 CBC

- New libraries.

3.22.1.3.3 ConvertID

- The GDX file created with option Jacobian contains now a set ANl that indicates the non-linear Jacobian elements.

3.22.1.3.4 CPLEX

- For an unbounded model Cplex did not mark the unbounded variables correctly. This has been fixed.

3.22.1.3.5 GUROBI

- Gurobi Optimization, Inc. has decided to end the current arrangement that allows us to offer GAMS/Gurobi integrated licenses. GAMS will continue to offer the latest version of Gurobi but users will need to get a Gurobi license from Gurobi Optimization, Inc. plus a GAMS/Gurobi link license from GAMS or use the free OsiGurobi link. This includes evaluation and demo licenses. For existing GAMS/Gurobi customers the following transition arrangements have been negotiated. The GAMS/Gurobi integrated license will continue to work as long as the license stays under maintenance. Academic GAMS/Gurobi customers will not be able to renew maintenance on their GAMS/Gurobi integrated license, their license will be changed into a GAMS/Gurobi link license. Academic users can get free Gurobi licenses from www.gurobi.com. In the past, academic GAMS/Gurobi link or GAMS/OsiGurobi licenses did not require a Gurobi license installed. This has changed with this release.
  - New libraries 6.5.1
  - The option PreSOS1BigM and PreSOS2BigM were incorrectly classified as integer options. They have been reclassified as real option.

3.22.1.3.6 IPOPTH

- Fixed a memory access issue in the linear solver HSL MA86 in case of very large models.

3.22.1.3.7 LocalSolver

- New libraries 6.0 (20160308).

3.22.1.3.8 MOSEK

- New libraries 7.1.0.49.
3.22.1.3.9 SBB

- Fixed a problem with option UserHeurCall. This option required that all discrete variables were declared first in the model. This is not necessary anymore.

3.22.1.3.10 SCIP

- New libraries 3.2 (#f69c505).
  - The timing mask for parameters constraints/.../presoltiming, presolving/.../timing, and propagating/.../presoltiming changed from 2/4/8 for fast/medium/exhaustive to 4/8/16.
  - Parameter constraints/SOS1/updateconflpresol has been renamed to constraints/SOS1/perfimplanalysis.
  - [Detailed Changelog](#).

3.22.1.3.11 SOPLEX

- New libraries 2.2 (#12d3858).

3.22.1.4 Tools

3.22.1.4.1 GAMS IDE

- Fixed a bug which caused an error when opening the FINLIB (Practical Financial Optimization Models).

3.22.1.4.2 MODEL2TEX

- Fixed a bug regarding negative variables.
- Fixed a bug regarding subscripts of symbol names with underscores.
- Fixed a bug that prevented alternative names from being replaced in Ord constructs.
- New parameter -o (-OUTPUT) that allows to specify an alternative output file.
- Use UTF8 encoding.
- The columns of the symbol table have a fixed (equal) width.
- The original name of a symbol that is changed using the JSON style file is displayed in the symbol table.
- The JSON style file is alphabetically sorted by keys.

3.22.1.5 Object Oriented APIs

- Fixed a bug with the property GAMSOptions.Defines (.NET), GAMSOptions.defines (Java), and GamsOptions.defines (Python): When too many entries were added, all of them were ignored.
3.22.1.5.1 Python

- Fixed a memory leak in GamsDatabase.

3.22.1.6 Model Libraries

3.22.1.6.1 GAMS Test Library

- scale01.gms : Test results of scaled model (692)

3.22.2 24.7.2 Minor release (July 07, 2016)

3.22.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Peter Alexander, Daniel Balistrieri, Paul Buckland, Denis Hess, Eden Huang, Josef Kallrath, Sherman Robinson, Tom Rutherford, Jhih-Shyang Shih, and Loic Ventre.

3.22.2.2 Platforms

- With the next major release, the minimal glibc version requirement for the GAMS distribution on Linux will be increased to 2.7.

3.22.2.3 GAMS System

3.22.2.3.1 GAMS

- Improved the performance for assignments from one large symbol to itself inside a loop, when only one element gets changed, e.g.:
  \[
  \text{loop}((j,cty,yrs,sim), \ldots; \text{par}(j,cty,'Total',yrs,sim) = \text{sum}(lnd, \text{par}(j,cty,lnd,yrs,sim)));
  \]
- Fixed a performance issue which caused very long model generation time for large scalar models.

3.22.2.4 Solvers

3.22.2.4.1 BARON

- New libraries 16.5.16.
  - Added option WantDual to indicate whether BARON should make a final call to an NLP solver to try to compute duals if an inexpensive method of calculating them failed.
- CompIIS option did not print IIS into listing file. This has been fixed.

3.22.2.4.2 CBC

- New libraries.
3.22.2.4.3 CPLEX

- New libraries 12.6.3.0.2.
- A memory leak in Cplex has been fixed that showed up in combination with the OO-API GAMSMo-
  dleInstance class.
- Fixed a problem with CplexD option FreeGamsModel.
- Fixed a problem with MipStart in combination with SemiInt or SemiCont variables with lower
  bound 0.

3.22.2.4.4 GUROBI

- New libraries 6.5.2.

3.22.2.4.5 JAMS

- Fix handling of objectives in optimizing agents of equilibrium systems in cases where the objective
  coefficient was not 1 or -1, and in the equation level returned for the objective row(s).

3.22.2.4.6 LocalSolver

- New libraries 6.0 (20160625).

3.22.2.4.7 Mosek

- New libraries 7.1.0.52.

3.22.2.4.8 OsiMosek

- Fixed handling of gap tolerances and node limit of MIP solves.

3.22.2.4.9 SBB

- Fixed a problem with models that have domain violations in the root NLP.
- Clearly identify text written to the listing file as output from the root solver.

3.22.2.4.10 SCIP

- New libraries 3.2 (#e99d344).

3.22.2.4.11 SoPlex

- New libraries 2.2 (#3c5e86f).
3.22.2.4.12  PATH

- Fixed a problem with bogus report of empty rows/cols when benchmarking a model, i.e. solving with iterlim=0.

3.22.2.4.13  XPRESS

- Several minor bug fixes that affect correctness and performance in some uncommon cases.

3.22.2.5  Tools

- Increase maximum available memory for GDXVIEWER, MDB2GMS, SQL2GMS, XLS2GMS, GDX2HAR, and HAR2GDX to 3GB.

3.22.2.5.1  CSV2GDX

- Fixed a problem with labels with trailing blanks. Labels with trailing blanks potentially resulted in a broken GDX file.
- Introduced the symbolic constant \texttt{LastCol} for the \texttt{Values} parameter.
- Report duplicate keys.

3.22.2.5.2  GDXXRW

- Fixed a problem with unnecessary memory consumption when writing with option clear or merge.
- Symbols and text were not written in sequence as instructed. This has been fixed.
- Added command line parameter \texttt{ReCalc} that by default prevents frequent recalculations in Excel while writing to the spreadsheet.

3.22.2.6  Object Oriented APIs

- Fixed a bug which did not clear a pending Ctrl-C event, so that it might have been applied to a following \texttt{GAMSModelinstance.Solve} by accident.
- Fixed a problem, that caused a crash instead of a \texttt{GAMSException} in some rare cases.

3.22.2.6.1  Python

- Fixed a minor bug regarding the names of GamsJob listing files.

3.22.2.7  Model Libraries

- Some URLs in various models have been updated.
3.22.3 24.7.3 Maintenance release (July 11, 2016)

3.22.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Josef Kallrath, and Toni Lastusilta.

3.22.3.2 Tools

3.22.3.2.1 CSV2GDX

- Fixed a bug which caused an error when reading sets from a csv file.

3.22.3.2.2 MODEL2TEX

- Fixed a bug regarding wrongly generated equations in the generated LaTeX file.

3.22.4 24.7.4 Minor release (September 19, 2016)

3.22.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Manuel Alvarez, Felix Färber, Ricardo Manuel Pinto de Lima, and Tara Rengarajan.

3.22.4.2 GAMS System

3.22.4.2.1 GAMS

- Fixed a problem when using extrinsic function libraries using API version 1.
- Extend possible range of \( n \) in \texttt{fact(n)} at compile time. It was limited to 20 in the past. Now it can be up to 166 (as in assignments at execution time).

3.22.4.2.2 Installer

- New default group name (\texttt{GAMS xx.x (xx bit)}) for the start menu on Windows 10.

3.22.4.3 Solvers

3.22.4.3.1 BARON

  - Small performance improvements and robustification of LP solver interfaces and bugfixes.
  - The starting point is now utilized before invoking projection and other initialization strategies.
  - The option \texttt{DeltaTerm} is now recognized again.
3.22.4.3.2 Couenne

- New libraries.

3.22.4.3.3 CPLEXD

- The option MipStart accepts now values other than 0 and 1. The value describes the effort level Cplex uses to determine a MipStart from the starting variable levels. The value 2 is interesting because it just checks for feasibility of the MipStart. In this case the level of all variables, not only the discrete ones, are passed on to Cplex.

3.22.4.3.4 DE

- DE crashed if multiple joint variables were present in one stage. This has been fixed.

3.22.4.3.5 Lindo/LindoGlobal

- New libraries 9.0.293.

3.22.4.3.6 Mosek

- New libraries 7.1.0.55.

3.22.4.3.7 SCIP

- New libraries 3.1 (#6b9196f).

3.22.4.3.8 SoPlex

- New libraries 2.2 (#074950a).

3.22.4.4 Tools

3.22.4.4.1 CSV2GDX

- Fixed a bug which lead to a rejection of unquoted labels with spaces. This bug was introduced in 24.7.2.

3.22.4.4.2 GDXXRW

- Fixed a bug which caused an error message when reading a sheet with a non-empty range that contained no data.
3.22.4.5 Object Oriented APIs

- `GAMSModelInstance.Instantiate`: Skip creation of GDX file, which was unreachable from within the APIs anyway.

3.22.4.5.1 Python

- Added support for Python 3.4 on Mac OS X.
- Fixed a problem with Exceptions in `GamsWorkspace`.

3.23 24.6 Distribution

3.23.1 24.6.1 Major release (January 18, 2016)

3.23.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Etienne Ayotte-Sauvé, Michael Ferris, Per Ivar Helgesen, Erwin Kalvelagen, Martha Loewe, Geoff Moore, and Renger van Nieuwkoop.

3.23.1.2 Platforms

- The minimal supported Mac OS X version has been increased to 10.9 (Mavericks). COIN-OR solvers, Gurobi, SCIP, and SoPlex now require a system with at least Mac OS X 10.9. The GAMS base system, tools, and other solvers are still working on Mac OS X 10.7 and 10.8, but may stop working in a future release without extra announcement.
- The installer for Wine on Linux has been dropped.

3.23.1.3 GAMS System

3.23.1.3.1 GAMS

- Increased controlled set nesting limit from 120 to 300.
- New option `SHUFFLE`.
- New option for command line parameter `MultiPass`:
  - 0: standard compilation
  - 1: check-out compilation
  - 2: as 1 and in addition skip `$call` and ignore errors because of missing files with `$include` (NEW)
- Allow to load variable and equation attributes into a parameter at compile time, e.g. `$LOAD par=var.L` (par being a parameter and var a variable)
Solving in parallel threads (beta feature)

A new variant for the SolveLink option and model attribute has been added. If SolveLink is set to 6 (compile time constant \%solveLink.Async Threads\%), GAMS does not wait for the solver to return and does not collect the result when a solve statement is executed. Instead, the model is generated and then passed to the solver in a separate thread while GAMS continues the execution. This way, multiple models can be solved in parallel and the results can be collected later.

This is similar to the Grid Facility (SolveLink = \%solveLink.Async Grid\%=3) with the difference, that the solver does not operate in its own process space but in a separate thread, which allows efficient in-memory communication between GAMS and the solver (like it is done with SolveLink = \%solveLink.Load Library\%=5). After the solve statement, one can store a handle of the model instance (using the model attribute mymodel.handle) and use the same functions that are used for the Grid Facility to collect the solution and deal with the model instance: HandleCollect(handle), HandleStatus(handle), and HandleDelete(handle).

The new function ReadyCollect(handleParameter [, maxWait]) can be used to wait until a model is ready to be collected. It can be used for both SolveLink = \%solveLink.Async Threads\% and SolveLink = \%solveLink.Async Grid\%. The arguments and return codes are:

- **Arguments**
  - handleParameter: parameter holding handles of model instances to wait for
  - maxWait: maximum time to wait (+INF if omitted)

- **Return Codes**
  - 0: (one of) the requested job(s) is ready
  - 1: no active job to wait for
  - 2: no handle provided
  - 3: invalid handle
  - 4: user specified time-out when using a SolveLink = \%solveLink.Async Threads\% handle
  - 5: user specified time-out when using a SolveLink = \%solveLink.Async Grid\% handle
  - 8: unknown error (should not happen)

The new option threadsAsync (available on the command line and with the option statement) sets the maximum number of threads that should be used for the asynchronous solves. If a negative number is set, this specifies how many of the available processors on the host machine should not be used. The default setting is -1.

Currently, the following solvers can be used with SolveLink = \%solveLink.Async Threads\%:

- CONOPT
- CPLEXD
- GUROBI
- MOSEK
- OSICPLEX
- OSIGUROBI
- SCIP

If another solver is selected, SolveLink = \%solveLink.Async Grid\% will be used instead (which is noted in the log).

An example of how this new feature can be used, can be seen in the GAMS Model Library model tgridmix.

### 3.23.1.3.2 Stochastic Programming with EMP

- Fixed a bug which caused wrong expected values for parametric distribution using the RandVar keyword.
3.23.1.4 Solvers

3.23.1.4.1 ANTIGONE

- New libraries.

3.23.1.4.2 BARON

- The handling of branching priorities in a GAMS/BARON options file has changed. Priorities should now always be given in the GAMS convention.

3.23.1.4.3 Convert

- GAMS branching priorities are now converted to BARON branching priorities when writing a BARON input file.

3.23.1.4.4 CPLEX

- New libraries 12.6.3.
- Cplex and CplexD now report the deterministic time spend (in ticks not in seconds) in the model attribute $\text{ETAlg}$.
- IBM's Cplex cloud offering "DOcloud" can be accessed via the Kestrel solver.

3.23.1.4.5 DICOPT

- New option $\text{usexinit}$ instructs DICOPT to start the NLP sub-solves from the user supplied input point.

3.23.1.4.6 GUROBI

- New libraries 6.5.
- New option $\text{PreMIQCPForm}$ that determines the format of the presolved version of an MIQCP model.
- New option $\text{WorkerPort}$ specifies a non-default port number for the distributed worker machines.
- New option $\text{VarHint}$. The variable hints communicated through $\text{level}$ and $\text{tryint}$ will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree.
- GAMS/Gurobi supports solving models in the Gurobi Instant Cloud.
- Fixed a problem with option $\text{IIS}$ for models with SOS variables.

3.23.1.4.7 IPOPT

- MKL Pardiso is now available as linear solver on Mac OS X, too.
3.23.1.4.8 KESTREL

- The Kestrel client provides experimental access to IBM's DOcloud offering.

3.23.1.4.9 PATH

- Minor fix for zero tolerance in the basis reset routine of the Lemke method.

3.23.1.4.10 Sulum

- Sulum will be dropped from the distribution with GAMS 24.8.

3.23.1.4.11 XPRESS

- Updated Optimizer libraries for Windows and Linux: 28.01.05 → 28.01.10.
- Several minor bug fixes that affect correctness and performance in some corner cases.

3.23.1.5 Tools

3.23.1.5.1 ASK

- New feature SelectDirectory.

3.23.1.5.2 GAMS IDE

- Sorting by symbol name is no longer case sensitive.
- GDX viewer can now show numbers with full precision.
- The option editor no longer shows the dot options.
- Fixed a bug where the cursor was not shown after double-clicking on a red line.

3.23.1.5.3 GDX2SQLITE

- New version 0.7.
  - Added option varchar to export character columns as VARCHAR(255) instead of TEXT.
  - Better names of columns when option small is used.
  - Adding timing info.

3.23.1.5.4 GDXMERGE

- Protect against very large symbols causing memory errors.
- Added option EXCLUDE to exclude symbols from being merged.
3.23.1.5.5 GDXXRW

- An empty range is no longer an error.

3.23.1.5.6 SQL2GMS

- In the old version double quotes were removed when reading a command file. In this version we keep double quotes to be able to escape SQL names (table names, column names). E.g. we now can handle non-standard names by double quoting them in a query. Depending on the database you can do:

  \[ q = \text{select "Some COLUMN" from "This Table"} \]

  Note that some databases (such as SQL Server) use [ ] for this goal.

  In the special case where you write:

  \[ q = \text{"select c from t"} \]

  the surrounding double quotes are removed before passing the query on to the database.

3.23.1.5.7 MODEL2TEX

- If an identifier is changed using the specifications in the JSON style file, underscores are no longer changed from "_" to "\_". This makes it possible to use subscripts when replacing an identifier.

3.23.1.6 Object Oriented APIs

3.23.1.6.1 Java

- Fixed a bug in `GAMSModelInstance.copyModelInstance` method when duplicating scratch directory.

3.23.1.6.2 Python

- New example `transport8a.py`.

3.23.1.7 Model Libraries

3.23.1.7.1 GAMS EMP Library

- `nbcontindep`: use sampling for continuous distributions if another solver than Lindo is selected
- `nbcontjoint`: use sampling for continuous distributions if another solver than Lindo is selected
- `nbsimple`: use discrete distribution
3.23.1.7.2 GAMS Test Library

- shuffle1.gms: Test for option shuffle (690)
- sl601.gms: Check correct behavior when using solveLink=6 (691)

3.24 24.5 Distribution

3.24.1 24.5.1 Major release (September 23, 2015)

3.24.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Guillaume Erbs, Michael Ferris, Mahbube Habibian, Josef Kallrath, Jean Mercenier, Stan Peter, Ingmar Schlecht, and Mark Thissen.

3.24.1.2 Platforms

- Support for Windows XP has been dropped completely with this release (as announced).

3.24.1.3 GAMS System

3.24.1.3.1 GAMS

- New model attributes
  - maxInfes: Maximum of infeasibilities
  - meanInfes: Arithmetic mean of infeasibilities
- New option solver. This simplifies the selection of the (default) solver for multiple model types.
  - The command line option solver=abc initializes the default solver for the model types solver abc is capable of to abc. This initialization is done before the default solvers of individual model types are set via command line options. So a command line with lp=conopt solver=bdmlp will first set BDMLP as the default solver for model types LP, RMIP, and MIP (these are the model types BDMLP can handle) and then reset Conopt as the default solver for LP. The order of these parameters on the command line has no impact (i.e. lp=conopt solver=bdmlp behaves identically to solver=bdmlp lp=conopt). If multiple occurrences of option solver appear, the last one sets the option as it is with other options, including LP, MIP, ...
  - The solver for multiple model types can be set via the option solver=abc; in the GAMS model source code. This sets the solver for model types abc can handle to abc. With the option solver=abc; the order among other solver setting options is significant. For example, option lp=conopt, solver=bdmlp; will first set the solver for LP to Conopt and in the next step to BDMLP because BDMLP is capable of handling model type LP. Setting solver twice can also make sense: option solver=conopt, solver=cbc; will result into setting the solver for model types CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt and the solver for model types LP, RMIP, and MIP to CBC.
- When using $LOADDC the reporting of domain errors has been improved.
- This release features several improvements of the execution system. The following lists some (extreme) examples:
– Improved performance when indices of accessed symbol are in "wrong" order, e.g.:

```gams
$if not set size $set size 10000
set v / v1*v%size% /
set e / e1*e%size% /;
alias(v,w);
set evw(e,v,w) /#e:#v:#w/;
vw(v,w) = sum(e, evw(e,v,w));
vw(v,w) = sum(e, evw(e,v,w));
The final assignment can be done in less than 1% of the time required by GAMS 24.4.
```

– Improved performance when assigning to non-empty symbols when the assignment is driven by their domain, e.g.:

```gams
$if not set size $set size 5000
set i / i1*i%size% /; alias (i,j);
parameter a(i,j), b(i,j);
a(i,j)$uniform(0,1)<0.95 = 1;
b(i,j) = not a(i,j);
a(i,j) = b(i,j);
The final assignment can be done in just \(~5\%\) of the time required by GAMS 24.4.
```

– Improved performance when executing certain combinations of mappings, e.g.:

```gams
$if not set size1 $set size1 8000
$if not set size2 $set size2 24
$if not set size3 $set size3 200000
set hi / 1*%size1% /
set ti / 1*%size2% /
set h / h1*h%size1% /
set t / t1*t%size2% /
hmapx(h,hi) / #h:#hi /
tmapx(t,ti) / #t:#ti /;
parameter xi(hi,ti), x1(h,t);
x1(h,t) = sum((hmapx(h,hi),tmapx(t,ti)), xi(hi,ti));
The final assignment can be done in just \(~2\%\) of the time required by GAMS 24.4.
```

– Improved performance when "searching" a lot in a large symbol, e.g.:

```gams
$if not set size1 $set size1 50
$if not set size2 $set size2 200000
$if not set size3 $set size3 150000
set f / f1*f%size1%/;
set j / j1*j%size2%/;
set l / l1*l%size3%/;
fjmap(f,l)=$uniform(0,1)<0.25;
parameter jpar(j,l) /#j:#l 10/
jpar(f,j) = sum(l$flmap(f,l), jpar(j,l));
The final assignment can be done in just \(~10\%\) of the time required by GAMS 24.4.
```

### 3.24.1.3.2 Documentation

- The complete GAMS technical documentation (release and installation notes, user's guides, solver and tools manuals, API reference manuals) is now available in HTML format with a unified table of contents. The documentation is available in the following forms:
  
  - The [online GAMS documentation](https://www.gams.com/) provides search and keyword indexing functionalities in addition to navigating the GAMS documentation.
3.24.1.3.3 Installer

- New way to install GAMS on Mac OS X using a DMG file.

3.24.1.4 Solvers

3.24.1.4.1 ANTIGONE

- Added option conopt_optfile to set name of options file to be used for CONOPT calls in ANTIGONE.

3.24.1.4.2 BARON

- New also available for Mac OS X.
- New libraries 15.9.22.
- New NLP solver for local search: FilterSD.
- Options:
  - New option Threads: Number of cores used for solution of MIP subproblems.
  - New option IISOrder: Order in which constraints are considered in the search for an IIS.
  - Options ConTol and IntTol removed. Tolerances are now set by AbsConFeasTol, RelConFeasTol, AbsIntFeasTol, and RelIntFeasTol.
  - Options MultMSize, MultRel, NLPDoLin, MipGap, MipNodes, MipRel, NoutIterMip, BilRel, Cvxbt, CvxInitOA, CvxRel, MipCuts, and RLTRel: Algorithmic features that have been parameterized are now handle in an automatic way.
- The time limit is enforced based on CPU time for single-threaded jobs and based on wall-clock time for multi-threaded jobs.
- Simplified the screen output (eliminated some details and simplified time to a real number in seconds).
- Updated CBC to 2.9.5.
- Updated Ipopt to 3.12.3.

3.24.1.4.3 CBC

- New libraries 2.9.
  - Introduced specialized branching methods for dealing with "big Ms".
  - Introduction of conflict cuts (off by default, enable with option conflictcuts).
  - Introduced new methods for dealing with symmetry.
3.24.1.4.4 Conopt

- New libraries 3.17A.
  - Corrected problem with options Ls2ndi and Lmusdf.
  - Systems errors related to very tight memory (984) and the inversion routine (2027) have been removed.
  - Three new options have been added to allow the modeler to see the order of the pre-triangular and post-triangular part of the model and the definitional equations:
    * PRPRET: Flag for printing the pre-triangular part of the model.
    * PRDEF: Flag for printing the defined variables and their defining constraints.
    * PRPOST: Flag for printing the post-triangular part of the model.

3.24.1.4.5 Couenne

- New libraries 0.5 (major update).
- Better handling of function signpower(x,k) for positive integer k ≤ 10.
- The default values for the following options have changed:
  - cont_var_priority from 2000 to 99.
  - int_var_priority from 1000 to 98.
  - feas_pump_milpmethod from -1 to 0.
  - feas_pump_poolcomp from 0 to 4.
  - output_level from 0 to 4.

3.24.1.4.6 CPLEX

- New option FreeGamsModel. This option preserves memory by dumping the GAMS model instance representation temporarily to disk. This option is available in CplexD only.

3.24.1.4.7 GUROBI

- New libraries 6.0.5 (technical release).
- New option FreeGamsModel. This option preserves memory by dumping the GAMS model instance representation temporarily to disk.

3.24.1.4.8 GUSS/ScenarioSolver

- The set of GUSS model attributes (Set ma "GUSS Model Attributes" / System.GUSSModelAttributes /;) has changed:
  - The attribute NumNOpt has been removed.
  - The attributes MaxInfes and MeanInfes have been added.

3.24.1.4.9 IPOPT

- New libraries 3.12 (minor changes).
3.24.1.4.10 JAMS

- **SubSolver** option fixed: it was sometimes ignored.

3.24.1.4.11 Kestrel

- The option file can be omitted. In this case, the Kestrel call is done with default settings.

3.24.1.4.12 Lindo/LindoGlobal

- Dropped Lindo/LindoGlobal libraries for Intel Solaris as announced previously.
- New libraries 9.0.225.

3.24.1.4.13 LocalSolver

- New libraries 5.5.
  - Improved accuracy and performance on numerical or mixed-variable optimization problems.

3.24.1.4.14 MINOS

- Adjusted default upper bound on the superbasics limit from 500 to 5000 and consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 5000 superbasics, limit the Hessian dimension to 5000 even though the superbasic limit is larger.

3.24.1.4.15 MOSEK

- New libraries 7.1.0.33.
- The GAMS option `iterlim` now sets the iteration limit for both simplex and interior point algorithms.
- More MOSEK options are now available in GAMS/Mosek:
  - `MSK_IPAR_MIO_CUT_CG`: Controls whether CG (Chvatal-Gomory) cuts should be generated.
  - `MSK_IPAR_MIO_CUT_CMIR`: Controls whether mixed integer rounding cuts should be generated.
  - `MSK_IPAR_MIO_PROBING_LEVEL`: Controls the amount of probing employed by the mixed-integer optimizer in presolve.
  - `MSK_IPAR_MIO_RINS_MAX_NODES`: Controls the maximum number of nodes allowed in each call to the RINS heuristic.
  - `MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS`: Maximum value of fractional part of right hand side to generate CMIR and CG cuts for.
  - `MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS`: Minimum value of fractional part of right hand side to generate CMIR and CG cuts for.
  - `MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT`: If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.
3.24.1.4.16 OS

- Now a hidden solver. Will be dropped from the distribution at some time (without further notice).

3.24.1.4.17 PATH

- New libraries 4.07.03.
- Adjusted to use wall-clock time on all platforms.
- Time limit checked more frequently, e.g. at each pivot.
- The presolve has been extended to find more reductions.
- Minor bug fixes.

3.24.1.4.18 SCIP

- New libraries 3.2 #c9c4375 (major update):
  - Presolving levels FAST, MEDIUM, and EXHAUSTIVE are now used to better coordinate the various presolvers.
  - Generalized upgrade from (SOC-representable) quadratic constraints to SOC constraints.
  - New primal heuristics “distribution diving”, “indicator”, and “bound”, improved clique and variable bound heuristics, and adjusted diving heuristics to solve fewer LPs.
  - New branching rules ”distribution”, ”multaggr”, and a new rule for SOS1 constraints.
  - New reliability notions and improved treatment of nonlinearities in hybrid reliability pseudo cost branching.
  - New separator ”eccuts” for generating edge-concave cuts for quadratic constraints and improved separation for convex quadratic constraints.
  - Decreased total memory usage by using more buffer data structures.
  - Improved propagation and separation for SOS of type 1 by using information from a conflict graph.
  - See also the full release notes, the change log, and the technical report.
- The following options were removed or replaced:
  - constraints/.../delaypresol and constraints/.../timemask replaced by constraints/.../presoltiming and constraints/.../proptiming.
  - presolving/domcol/singcolstuffing replaced by presolving/stuffing/....
  - presolving/.../delay replaced by presolving/.../timing.
  - propagating/.../presoldelay replaced by propagating/.../presoltiming.
  - propagating/obbt/maxlookahead removed.
- For the following options, the default value changed:
  - constraints/SOS1/sepaq frec from 0 to 10.
  - ”heuristics/clique/minfixingrate” from 0.5 to 0.25.
  - ”heuristics/vbounds/minfixingrate” from 0.5 to 0.25.
  - heuristics/actconsdiving/maxdiveavgquotnosol from 0 to 1.
- heuristics//actconsdiving/maxdiveubquotnosol from 0.1 to 1.
- heuristics/dins/nwaitingnodes from 0 to 200.
- lp/rowrepswitch from -1 to 1.2.
- presolving/abortfac from 0.0001 to 0.001.
- presolving/restartfac from 0.05 to 0.025.
- presolving/immrestartfac from 0.2 to 0.1.
- presolving/dualinfer/priority from 20010000 to -200.
- propagating/obbt/itlimitfactor from 5 to 10.

3.24.1.4.19 SoPlex

- New libraries 2.2.0 (major update).

3.24.1.4.20 SULUM

- New libraries 4.3.892.
  - Several bug fixes in both the MIP solver and the LP solver.
  - Improved numeric stability, degeneracy handling, and perturbation scheme in LP optimizer.
  - Added a new combined pricing scheme to the dual simplex optimizer.
  - Improved presolve and restart in MIP optimizer.

3.24.1.4.21 XPRESS

- New libraries for XPRESS v7.9: Optimizer 28.01.05.
  - Significantly improved linear algebra routines for the simplex solvers improving efficiency of a wide range of problems.
  - Improvement heuristics called more often.
  - The MIP log now provides information which heuristic finds a solution.
  - Improved linear dependency checker for large problems.
  - Improved scaling, including scaling of big-M type rows and of the Curtis-Reid scaling option.
  - Improved inference-learning from infeasible subproblem during the MIP search.
  - Improved presolver for quadratic instances.
  - Improved handling of Special Ordered Sets.
  - Improved propagation of conflict cuts.

3.24.1.5 Tools

3.24.1.5.1 GAMS IDE

- In the model library browser the IDE may provide hints about the content of the column when hovering over the headers.
- Option to specify file extensions for files that will be reloaded without a confirmation dialog.
3.24.1.5.2 GDXDIFF

- When renaming the temporary file to `gdxdiff.gdx` fails, issue a ViewClose command in case the file is open in the GAMSIDE and try to rename again.
- If no difference was found, issue a message in the log indicating this.
- Add a set with two elements with explanatory text of the two files compared.

3.24.1.5.3 GDXMERGE

- When a filename cannot be used as a UEL, use a generated name instead.

3.24.1.5.4 GDXXRW

- New options Ignor erows and IgnoreColumns to ignore a set of rows or columns for a symbol. Rows can be specified as `IgnoreRows=1,4:5` and columns `IgnoreColumns=A,D:F` or `IgnoreColumns=1,4:6`.

3.24.1.5.5 MODEL2TEX

- The beta version of this tool allows the automatic generation of LaTeX code that documents a given GAMS model.

3.24.1.5.6 MPSGE

- The documentation of MPSGE has moved again. It can now be found in the User’s Guide.

3.24.1.6 Object Oriented APIs

- New examples `SpecialValues` and `Clad` (Java: `specialvalues/SpecialValues.java, clad/Clad.java`; Python: `special_values.py, clad.py`).
- New functions to retrieve models from the GAMS API Library and the Nonlinear Optimization Applications Library:
  - Java: `GAMSWorkspace.addJobFromApiLib` and `GAMSWorkspace.addJobFromNoaLib`.
  - Python: `GamsWorkspace.apilib` and `GamsWorkspace.noalib`.

3.24.1.6.1 Python

- Added support for Python 3.4 (Windows and Linux only). The examples have been changed to be compatible with all supported Python versions.
3.24.1.7 Model Libraries

3.24.1.7.1 NOALIB - Nonlinear Optimization Applications Using the GAMS Technology

- This new library by Neculai Andrei has been added to the GAMS system. This is a collection of the models based on the book *Nonlinear Optimization Applications Using the GAMS Technology* by Neculai Andrei. The library contains a wide spectrum of nonlinear optimization applications expressed in GAMS. The book and library emphasize the local solutions of the large-scale, complex, continuous nonlinear optimization applications, and the abundant examples in GAMS are highlighted by those involving ODEs, PDEs, and optimal control. The collection of these examples will be useful for software developers and testers.

- You can retrieve the individual models through the IDE model library browser or via the command line utility `noalib`.

3.24.1.7.2 GAMS API Library

- CSSpecialValues.gms: Test handling of Special Values in object oriented C# API
- PSpecialValues.gms: Test handling of Special Values in object oriented Python API
- JSpecialValues.gms: Test handling of Special Values in object oriented Java API
- CSNUnit.gms: Compiles and runs NUnit tests for object oriented C# API
- CSClad.gms: Test changing solver options while running using the interrupt method
- PClad.gms: Test changing solver options while running using the interrupt method
- JClad.gms: Test changing solver options while running using the interrupt method

3.24.1.7.3 GAMS Data Library

- `invert1`: Pass `gams.sysdir%` to R script to make linkage between GAMS and R explicit.
- `gdxmrw_tr3`: Canonical form LP created in Matlab and solved via `gams()` Mex-function.
- `gdxmrw_tr4`: Better example of `gams()` usage.
- `gdxmrw_tr5`: Better example of `gams()` usage.

3.24.1.7.4 GAMS EMP Library

- `transecs`: Fix formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.
- `farmnbd.gms`: The Farmer's Problem - Stochastic with NBD
3.24.1.7.5  GAMS Model Library

- In the IDE the GAMS Model Library browser has now a Lic column indicating the license requirement of a model. The letters D and G indicate that the model does not require a license. Models with G can even be solved by a global solver without a license (the demo limit for global solvers is 10 variables and 10 equations). A letter L indicates that a license is required.
  - tgridmix: Fix the logic for sleeping. Only do that if no more jobs to be scheduled or all cores are busy.

3.24.1.7.6  GAMS Test Library

- asynntrp: Make this model work under Unix that have the pstree utility available.
- call6.gms: Call GAMS in a folder containing a %
- cmexc01.gms: Trigger unexpected cmexRC error
- rs02.gms: Solving Three-dimensional Noughts and Crosses using Cplex and Gurobi distributed MIP
- single04.gms: Check handling of singleton sets assigned and referenced in a loop
- model2tex1.gms: Test that model2tex produces a tex file
- exmcp6.gms: External Equation - Example MCP 6
- scensol6.gms: Test execute_loadhandle for GUSS/GRID
- idxperm1.gms: Check correct behavior when permuting indices in model generation
- idxperm2.gms: Check correct behavior when permuting indices in loop etc

3.24.2 24.5.2 Maintenance release (September 29, 2015)

3.24.2.1  Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Kevin Doran and Renger van Nieuwkoop.

3.24.2.2  GAMS System

3.24.2.2.1  GAMS

- Fixed a bug which caused a crash for certain assignments, in particular it had to be an assignment to a symbol that was also referenced on the RHS, with permuted indices and we actually set some of the records to zero which were non-zero before.

3.24.2.3  Object Oriented APIs

3.24.2.3.1  .NET

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. (This is for Mono only.)
3.24.2.3.2 Python

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. The property `GamsWorkspace.system_directory` now always returns the canonical path with all symbolic links resolved.

3.24.2.4 Model Libraries

3.24.2.4.1 GAMS Test Library

- `idxperm3.gms`: Check correct behavior when permuting indices of symbol used on LHS and RHS (686)

3.24.3 24.5.3 Maintenance release (October 01, 2015)

3.24.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz.

3.24.3.2 GAMS System

3.24.3.2.1 GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments

3.24.4 24.5.4 Maintenance release (October 15, 2015)

3.24.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Per Ivar Helgesen and Jan Imhof.

3.24.4.2 GAMS System

3.24.4.2.1 GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments, in particular it had to be an assignment using the same controlling set more than once in one symbol, e.g. `p(1, j, 1)`.
- Fixed a bug when handling suffixes (e.g. `.val`) of Singleton Sets in Equations.

3.24.4.3 Object Oriented APIs

3.24.4.3.1 .NET

- Fixed a bug with the property `GAMSOptions.IDir`.
3.24.4.4 Model Libraries

3.24.4.4.1 GAMS Test Library

- single05.gms: Check correctness of set attributes for singleton sets (687)
- idxperm4.gms: Check correct behavior when permuting indices of symbol using same controlling set multiple times (688)

3.24.5 Maintenance release (November 25, 2015)

3.24.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Albrecht, Stefan Kemnitz, Martha Loewe, Thomas Maindl, Bruce McCarl, and Andres Ramos.

3.24.5.2 GAMS System

3.24.5.2.1 GAMS

- GAMS now always reports the upper bound that is sent to the solver in the column listing and solution report. Hence, with IntVarUp=1 GAMS prints the value of 100 for integer and semiint variables that are at default bound value +INF.
- Fixed a bug which could cause a crash when referencing attributes of equations (e.g. marginals) after a model was solved with a non-default solveLink setting and there was no solution returned.

3.24.5.2.2 Stochastic Programming with EMP

- Fixed a bug which caused wrong sample sizes if the sample keyword was used more than once and with different sample sizes.

3.24.5.3 Solvers

3.24.5.3.1 Cbc

- New libraries.

3.24.5.3.2 CplexD

- Fixed a problem with the computeserver option. The value for the computeserver option is limited to 255 characters. In order to specify a longer list of workers, one can now use multiple lines with the computeserver option.
3.24.5.3.3 Ipopt

- New libraries.

3.24.5.3.4 LocalSolver

- New libraries 5.5 (20151028).

3.24.5.3.5 Mosek

- New libraries 7.1.0.41.

3.24.5.3.6 SCIP

- New libraries 3.2 (#e9a5ca7).
  - Removed options constraints/SOS1/bipbranch, constraints/SOS1/neighbranch, and constraints/SOS1/sos1branch.

3.24.5.3.7 SoPlex

- New libraries 2.2 (#f17b9e7).

3.24.5.4 Tools

3.24.5.4.1 put_toexcel, put_tohtml

- Fixed a problem with put_toexcel and put_tohtml.

3.24.6 24.5.6 Maintenance release (November 27, 2015)

3.24.6.1 Solvers

3.24.6.1.1 COUENNE

- New libraries.

3.24.6.1.2 IPOPT, BONMIN, SCIP

- Fixed issue that Ipopt and Bonmin always read ipopt.opt, despite of the optfile setting in GAMS.
3.25 24.4 Distribution

3.25.1 24.4.1 Major release (December 20, 2014)

3.25.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Jeff Linderoth, and Erkka Rinne.

3.25.1.2 GAMS System

3.25.1.2.1 GAMS

- Fixed a bug causing a potential crash when solving an EMP model having an open file handle without an external file name at the same time.
- The execMode setting is not ignored by put utility anymore.
- Added gbin/md5sum.exe to Windows distribution. This utility allows users to verify the file integrity of newly downloaded GAMS installation files.
- The gbin_new directory will be removed with the next distribution.

3.25.1.2.2 Installer

- Windows installer removes the GAMS entry in the current user hive of the registry, if GAMS is installed for all users. Previously, the hive for the current user remained unchanged in this situation.

3.25.1.2.3 Extrinsic Function libraries

- The new extrinsic function library parcclib was added to the system. This library demonstrates how to access the GAMS parameter file and use it's information through the GAMS Option Object. Further information can be found in the GAMS User's Guide, Appendix J.
- Allow up to 20 arguments (used to be 10).
- Support for extrinsic function libraries that do not provide gradient and/or Hessian values. GAMS uses finite differences (same method as for .gradn and .hessn) to approximate the derivatives, even inside a solver. The example function library trilib now implements function Sine without any derivatives and leaves their calculation to GAMS.
- New GAMS options have been introduced to parameterize the numerical derivative calculations. While these are most important for extrinsic functions, they also apply to the .gradn/.hessn calculations of intrinsic functions.
  - FDDelta: step size in the numeric gradient and Hessian calculation. For single argument functions, GAMS evaluates the function at f(x-d) and f(x+d) for the numerical gradient. If function values are used for the numerical Hessian, GAMS evaluates at f(x-2d), f(x), and f(x+2d). For multi argument functions, the same is done for the components of the input argument vector. The default for FDDelta is 1e-5. This option can be set from the command line, in an option statement, and as a model attribute.
  - Note: In previous releases it was possible to set the step size for .gradn/.hessn using the option Real1. This does not work anymore.
3.25 24.4 Distribution 261

- **FDOpt**: The option value packs two option in the different digits: $ij$. The $i$ digit indicates if scaling of the step size ($\text{FDDelta}$) by the value of the input argument should be turned off. If $i=0$ (scaling on) the following step size is used: $\max(1,|x|)\times\text{FDDelta}$. The $j$ digit is mostly for testing, but has one setting that might be relevant when the extrinsic function provides gradient but no Hessian values. The numerical derivatives routine in this case uses the gradient calculation from the extrinsic function to approximate the Hessian. If the gradient is expensive to calculate compared to a function evaluation, it could be beneficial to use multiple function values to approximate the Hessian. In this case set the $j$ digit to 1. Here are all possible values for this option:
  * 0: All derivatives analytically if available, for numerical Hessian use gradient values, scale delta
  * 1: All derivatives analytically if available, for numerical Hessian use function values, scale delta
  * 2: Gradient analytically, force Hessian numerically using gradient values, scale delta (testing only)
  * 3: Gradient analytically, force Hessian numerically using function values, scale delta (testing only)
  * 4: Force gradient and Hessian numerically, scale delta (testing only)
  * 10: Same as 0, but no scale of delta
  * 11: Same as 1, but no scale of delta
  * 12: Same as 2, but no scale of delta (testing only)
  * 13: Same as 3, but no scale of delta (testing only)
  * 14: Same as 4, but no scale of delta (testing only)
- **Note**: In previous releases it was possible to deactivate the scaling for `.gradn/.hessn` using the option `Integer1`. This does not work anymore.

- Renamed and better documented the `EXTRFUNC_ERROR` enum to `EXTRFUNC_EVALERROR` in the C header `extrfunc.h`. The old values are deprecated and will be removed in a future release.

### 3.25.1.2.4 Documentation

- The manuals of all solver and several tools are now available in HTML format. A PDF that contains all solver manuals is still available (`docs/solvers/allsolvers.pdf`).

### 3.25.1.3 Solvers

#### 3.25.1.3.1 AlphaECP

- Alpha-ECP v2.10.06.
  - Added support for solver trace file, i.e. new option `solvetrace`.

#### 3.25.1.3.2 ANTIGONE, GloMIQO

- Now also available for Mac OS X.

#### 3.25.1.3.3 BARON

- New libraries 14.4.0.
  - Added reliability branching for integer variables.
  - Bug fixes in CBC interface and IIS module.
  - Improved performance of problem reading and reformulation.
  - Updated Ipopt to version 3.11.9.
3.25.1.3.4 BENCH

- Removed option `cumulative`. This option makes no sense anymore since all solvers are spawned asynchronously and all get the same limits.

3.25.1.3.5 CBC

- Fixed a race-condition in log output when using multiple threads.
- New libraries.

3.25.1.3.6 Couenne

- New libraries.

3.25.1.3.7 CONOPT

- New libraries 3.16D.
  - Tolerance adjustments that make sense if NaN appears in intermediate calculations.

3.25.1.3.8 CPLEX

- Don't rerun primal simplex in cases where there is already an infeasibility status.
- New libraries 12.6.1.
- New parameters:
  - `qtolin` controls the linearization of the quadratic terms in the objective function of a QP or MIQP model
  - `localimplied` controls the generation of locally valid implied bound cuts

3.25.1.3.9 DICOPT

- Added support for `=N=` rows.

3.25.1.3.10 GUROBI

- New libraries 6.0.0.
- Fixed problem in tuning. Previous version did not write out link options in the tuned option file.
- Gurobi 6.0 supports a distributed MIP algorithm (option `DistributedMIPJobs`). This requires the Gurobi Compute Server to be licensed.
- Explicit lazy constraint: Users can use the "dot" option `.lazy` to mark constraints as lazy. Possible values are 0, 1, 2, and 3. See option `.lazy` for details.
- Option `ConcurrentMIPJobs` has been renamed to `ConcurrentJobs`. The distributed concurrent optimizer now also handles LP models.
- Option `ScaleFlag` allows now value 2 which enables aggressive scaling.
- New parameters:
  - `DistributedMIPJobs` controls the number of workers of the distributed MIP algorithm
  - `.lazy` marks constraints as lazy constraints
  - `LazyConstraints` controls the use of lazy constraints
3.25.1.3.11 Knitro

- New libraries 9.1.0.
  - Overall speed and robustness improvements on NLP and MINLP models.
  - New algorithm choice introduced: active-set SQP.
  - Optional barrier solution refinement procedure: option bar_refinement = NO/yes.
  - Deterministic parallel multi-start: option ms_deterministic = no/YES.
- Dropped Knitro for Solaris on Intel CPUs (as announced).

3.25.1.3.12 LocalSolver

- Added the hybrid mathematical programming solver LocalSolver 5.0 (Beta) from Innovation 24 to the GAMS solvers portfolio.
- LocalSolver can be applied to large-scale, mixed-variable, non-convex optimization problems (GAMS model types MIP, (MI)QCP, (MI/D)NLP). It combines local search techniques, constraint propagation and inference techniques, linear and mixed-integer programming techniques, and nonlinear programming techniques in a unique hybrid neighborhood search approach to find high-quality feasible solutions. Hence, LocalSolver offers an alternative for problems where conventional branch-and-bound and/or outer-approximation based solution methods do not provide satisfactory results.
- It is suggested to set the GAMS options for iteration or timelimit (iterlim, reslim) appropriately to limit the effort that LocalSolver spends on the problem.

3.25.1.3.13 Lindo/LindoGlobal

- We will drop Lindo/LindoGlobal libraries for Intel Solaris with the next major release (24.5).
- New libraries 9.0.142 for Linux, Mac OS X, and Windows:
  - Support for semi-continuous variables.
  - Simplex LP algorithm implementation has been improved for speed and robustness. The performance improvements compared to previous version are 90% for primal simplex and 45% for the dual simplex.
  - Knapsack related cuts improvements. Significantly faster solve times on models with certain knapsack-like constraints.
  - Improved default node selection rules improves performance on most MIPs.
  - New branching variable rule options: maximum coefficients and neighborhood branching. Can reduce number of branches on certain MIPs.
  - Perspective reformulation capability gives improved performance on quadratic portfolio models with semi-continuous variables, e.g. min-buy quantities.
  - Improved default settings for NLPs gives 5% average speed improvement.
  - New preprocessing for LP/MIP significantly reduces coefficient density of certain dense matrices.

3.25.1.3.14 Mosek

- New libraries 7.1.0.12.
  - Improved performance of the mixed-integer conic optimizer.
3.25.1.3.15 MSNLP/OQNLP

- These solvers now use Conopt as their default solver if Conopt is licensed. Otherwise they will use lsgrg as before.

3.25.1.3.16 OS

- We plan to drop the OS solver with GAMS 24.5. The capability of convert to write OSiL files will be kept.

3.25.1.3.17 SCIP

- New libraries 3.1 #020e055.

3.25.1.3.18 SoPlex

- New libraries 2.0 #d67b17b.

3.25.1.3.19 Sulum

- New libraries 4.0.665.
  - Improvements in presolve, leading to more reductions.
  - General improvements in obtaining a feasible solution faster, especially with focus on the root node to obtain a good bound. Rewrite and improvements of feasibility pump, objective diving.
  - The simplex algorithm is now exchanging more information with the branch and cut method. Improvement of SINS heuristic (finding a better basis after the node solve).
  - Added a MIP root restart feature based on reductions.
  - Cutting planes generation was both improved and extended.
  - Changed default for option `mipmaxrestarts` from 5 to 1.

3.25.1.3.20 XPRESS

- New libraries for XPRESS v7.8: Optimizer 27.01.02.
  - Automatic solution refinement for LP and MIP models.
  - Improved deterministic concurrent LP.

3.25.1.4 Tools

3.25.1.4.1 GAMSIDE

- Added option to suppress trailing zeroes in GDX viewer.
- Improved ability to open files from the Windows Shell.
- Cutoff for number of lines to be syntax colored or not.
- When saving a file, the Undo buffer is no longer cleared.
3.25.1.4.2 GDXDUMP

- Text quoted with a single quote did not have a separator when writing SymbolAsSet.
- Added option SymbolAsSetDI.
- Avoid string overflow when quoting text.
- Added more types for sets and equations.

3.25.1.4.3 GDXXRW

- Added option values=All which is the new default when CDim=0 or RDim=0.

3.25.1.4.4 GDXVIEWER

- Added ACCDB format as an output option for MSAccess.

3.25.1.4.5 GDX2SQLite, Scenred, Scenred2

- These tools are now also available for AIX.
- Moved documentation of Scenred and Scenred2 from docs/solvers to docs/tools.

3.25.1.4.6 IDECMDS

- Allow a pattern to be specified to close files.

3.25.1.4.7 MPS2GMS

- The mps2gms tool now produces proper generic GAMS source for models with quadratic terms.
- Bug fix for MPS files written by Cplex that contain SOS constraints.

3.25.1.4.8 MPSGE

- Moved manual to "docs/tools".

3.25.1.5 Expert Level APIs

- There are new API files for the "Indexed GDX" (IDX) library in $GAMS Dir$/apifiles$/Language$/api:
  - The indexed GDX library can be used to read and write indexed GDX files.
  - Each symbol in such a GDX file must be a parameter.
  - Each parameter must have a domain consisting of a UELs which forms an integer sequence starting at 1.
  - This way the data is provided in a format convenient to store in arrays in the target language.
  - In GAMS such a file can be read using $LoadIDX$ during compilation.
  - In GAMS such a file can be written using $execute_unloadIDX$ during execution.
  - The new example $GAMS Dir$/apifiles$/CSharp$ $xp_CalcInverseIDX$ makes use of this API, this example is also used in the APIlib model $CSCalcInverse$. 
3.25.1.5.1 Python

- Fixed a bug regarding lists of strings in Python 3 (e.g. `gdxDataWriteStr()`).

3.25.1.6 Object Oriented APIs

- We changed the handling of GAMS Aliases in the object oriented APIs:
  - If we ask for the number of GAMSSymbols in a GAMSDatabase, the Aliases will be excluded.
  - If we iterate over all GAMSSymbols in a GAMSDatabase, Aliases will be skipped.
  - If we ask explicitly for an Alias in a GAMSDatabase (`GAMSDatabase.GetSet("a")` with `a` being an Alias) we will get a reference to the GAMSSet referenced by the Alias, not the Alias itself.
  - Note: Aliases can appear in a GAMSDatabase only, if it was initialized by a GDX file containing an Alias.
  - The new examples `Alias` demonstrate this new behavior for the different OO API languages.

3.25.1.6.1 Python

- Fixed a bug in `GamsDatabase.merge_record` that prevented the function from creating a record if none was found.

3.25.1.7 Model Libraries

3.25.1.7.1 GAMS API Library

This is a new collection of GAMS models. It can be accessed in the GAMS IDE at Model Libraries -> GAMS API Library or through the command line tool `apilib`. The models in this collection can be used as scripts to compile and execute the example applications using the GAMS object oriented APIs as well as the expert level APIs, which can be found in `<GAMS Dir>/apifiles`.

- `apiutil.gms`: Generates the API Model Library files (01)
- `testapi.gms`: API Quality Assurance Test (02)
- `Cex1.gms`: Test expert level C API to read and write GDX (03)
- `CSex1.gms`: Test expert level C# API to read and write GDX (04)
- `CPPex1.gms`: Test expert level C API to read and write GDX in C++ (05)
- `DCex1.gms`: Test expert level Delphi (function) API to read and write GDX (06)
- `DOex1.gms`: Test expert level Delphi (object) API to read and write GDX (07)
- `DPex1.gms`: Test expert level Delphi (pure) API to read and write GDX (08)
- `Fex1.gms`: Test expert level Fortran API to read and write GDX (09)
- `Jex1.gms`: Test expert level Java API to read and write GDX (10)
- `Pex1.gms`: Test expert level Python API to read and write GDX (11)
- `VBex1.gms`: Test expert level VB.Net API to read and write GDX (12)
- `Cex2.gms`: Test expert level C API to read and write GDX, set options and execute GAMS (13)
• CSex2.gms: Test expert level C# API to read and write GDX, set options and execute GAMS (14)
• CPPex2.gms: Test expert level C API to read and write GDX, set options, and execute GAMS in C++ (15)
• DOex2.gms: Test expert level Delphi (object) API to read and write GDX, set options and execute GAMS (16)
• Fex2.gms: Test expert level Fortran API to read and write GDX, set options and execute GAMS (17)
• Jex2.gms: Test expert level Java API to read and write GDX, set options and execute GAMS (18)
• Pex2.gms: Test expert level Python API to read and write GDX, set options and execute GAMS (19)
• VBex2.gms: Test expert level VB.Net API to read and write GDX, set options and execute GAMS (20)
• CStrseq.gms: Test object oriented C# API running a sequence of examples based on a transport model (21)
• CPPtrseq.gms: Test object oriented C++ API running a sequence of examples based on a transport model (22)
• Jtrseq.gms: Test object oriented Java API running a sequence of examples based on a transport model (23)
• Ptrseq.gms: Test object oriented Python API running a sequence of examples based on a transport model (24)
• VBtrseq.gms: Test object oriented VB.Net API running a sequence of examples based on a transport model (25)
• CSBenders.gms: Test object oriented C# API using a (multi-threaded) Benders Decomposition Algorithm (26)
• JBenders.gms: Test object oriented Java API using a (multi-threaded) Benders Decomposition Algorithm (27)
• PBenders.gms: Test object oriented Python API using a (multi-threaded) Benders Decomposition Algorithm (28)
• CSCutstock.gms: Test object oriented C# API using a cutting stock example (29)
• JCutstock.gms: Test object oriented Java API using a cutting stock example (30)
• PCutstock.gms: Test object oriented Python API using a cutting stock example (31)
• CSDomainChecking.gms: Test object oriented C# API for domain checks (32)
• JDomainCheck.gms: Test object oriented Java API for domain checks (33)
• PDomainChecking.gms: Test object oriented Python API for domain checks (34)
• JInterrupt.gms: Test object oriented Java API for interrupting running GAMS jobs (35)
• CSTsp.gms: Test object oriented C# API using a Traveling Salesman Problem (36)
• JTsp.gms: Test object oriented Java API using a Traveling Salesman Problem (37)
• PTsp.gms: Test object oriented Python API using a Traveling Salesman Problem (38)
• CSWarehouse.gms: Test object oriented C# API using a warehouse location problem (39)
• JWarehouse.gms : Test object oriented Java API using a warehouse location problem (40)
• PWarehouse.gms : Test object oriented Python API using a warehouse location problem (41)
• CSAlias.gms : Test handling of Aliases in object oriented C# API (42)
• JAlias.gms : Test handling of Aliases in object oriented Java API (43)
• PAlias.gms : Test handling of Aliases in object oriented Python API (44)
• apihtm.gms : Generates HTM apilib library files (45)
• CSCalcInverse.gms : Test expert level C# API to read and write indexed GDX (46)

3.25.1.7.2 GAMS Data Library

• Note: Opening an Excel file (.xls) in protected view may work improperly due to Microsoft issue 2745652. Fix: Enable editing of the Excel file and reopen the file.
• gdxmrw_qp1_starter.gms : Portfolio Analysis with Matlab and GAMS (91)
• gdxmrw_qp2_starter.gms : Portfolio Analysis with Matlab and GAMS (92)
• gdxmrw_tr1.gms : Transport LP with non-indexed GDX data interface (93)
• gdxmrw_tr2.gms : Transport LP with indexed GDX data interface (94)
• gdxmrw_qp3.gms : QP solver M-file using GAMS and GDXMRW (95)
• gdxmrw_autoopen.gms : Tests that gdxrxw calls the auto_open macro facility (96)
• gdxmrw_qp4.gms : Calling GAMS model from Matlab (97)
• gdxmrw_intro01_init.gms : Introduction to data transfer between Matlab and GAMS (98)
• gdxmrw_intro02_init.gms : Introduction to calling GAMS from Matlab (99)
• gdxmrw_irgdx01_init.gms : Reading data from a indexed GDX file with IRGDX (100)
• gdxmrw_iwgdx01_init.gms : Writing data into a indexed GDX file with IWGDX (101)
• gdxmrw_rgdx01_init.gms : Reading data from a GDX file into a structure with RGDX (102)
• gdxmrw_wgdx01_init.gms : Writing structured data into a GDX file with WGDX (103)
• gdxmrw_ext01_init.gms : Extended use of GDXMRW (104)

3.25.1.7.3 GAMS Model Library

• Modified models
  – tsp1 and tsp4 now use Singleton Sets
  – flowchan had incorrect boundary conditions
  – licememo: give a full solver/model type matrix independent of the actual license
• partssupply.gms : Parts Supply Problem (404) (contains ps2f,s .. ps5_s,mn, which are still available, as submodels)
• qfilter.gms : Audio filter design using quad-precision MINOS (405)
• derivtst.gms : How to test derivatives of functions (406)
• carseq.gms : Car Sequencing (407)
• pmedian.gms : P-Median problem (408)
• sgolfer.gms : Social Golfer Problem (409)
3.25.1.7.4 GAMS Test Library

- Modified Models
  - trilib01, trilib02, and trilib03: Compare numeric gradient and Hessian values in the models. Function Sine in the library does not provide derivatives anymore. Lower the tolerances for acceptance due to numerical derivatives of function Sine

- indic04.gms : Test of indicator constraints with explicit labels (663)
- parlib01.gms : Test extrinsic functions in parcclib (667)
- convert10.gms : CONVERT test suite - check interval evaluator in Convert (668)
- convert11.gms : CONVERT test suite - check interval evaluator in Convert (669)
- convert12.gms : CONVERT test suite - check interval evaluator in Convert (670)
- lazy01.gms : Test lazy constraints (671)
- mps2gms1.gms : Test mps2gms (672)
- execmode01.gms : Test execmode behavior (673)

3.25.2 24.4.2 Minor release (March 15, 2015)

3.25.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Sergey Kuznetsov, Ricardo M. De Lima, and Corey Noone.

3.25.2.2 GAMS System

3.25.2.2.1 GAMS

- Fixed potential problem when running on Windows in a workDir containing a %.
- Fixed problem with EMP solves with solvelink=Solvelink.AsyncGrid%.
- Fixed problem with GUSS/Grid when using execute_loadhandle instead of handlecollect.

3.25.2.3 Solvers

3.25.2.3.1 ANTIGONE

- New libraries.

3.25.2.3.2 CONOPT

- Introduced new boolean option PreTri2Log that send message from Conopt's pre-triangular analyzer that go to the listing file also to the GAMS log. The option requires the generation of the model dictionary, so mymodel.DictFile=1; has to be added before the Solve statement.
3.25.2.3.3 CONVERT

- Convert and ConvertD: Fix for writing scalar MCP models that contained fixed variables removed by \texttt{holdfixed=1}.
- ConvertD: Add support for external equations in scalar GAMS models.
- ConvertD: Propagate EPS to scalar GAMS models.

3.25.2.3.4 Couenne

- New libraries.

3.25.2.3.5 Examiner2

- Support added for custom trace files.

3.25.2.3.6 GUROBI

- New libraries 6.0.2.

3.25.2.3.7 Lindo/LindoGlobal

- New libraries 9.0.157 for Linux, Mac OS X, and Windows.

3.25.2.3.8 LocalSolver

- New libraries 5.0 (20150119).

3.25.2.3.9 Mosek

- New libraries 7.1.0.24.

3.25.2.3.10 SCIP

- New libraries 3.1 \#67d713c.
- Fixed overwriting of solvetrace file by final NLP resolving.

3.25.2.3.11 SoPlex

- New libraries 2.0 \#8381aa4.
3.25.2.4 Tools

3.25.2.4.1 GDXMRW

- gdxInfo: fix output of domains in symbol declaration
- gdxInfo: handle aliases properly

3.25.2.5 Object Oriented APIs

- Fixed overwriting of the default value for the integer1 option when running with GAMSModelInstance, which lead to unexpected solver behavior.
- Fixed potential problem with GAMSModelInstance used with BARON.

3.25.2.5.1 .NET

- New property GAMSSymbol.DomainsAsStrings: Domains of Symbol, each element is a string, if the domain is an alias in GAMS, this call will return the name of the Alias, not the name of the aliased Set.

3.25.2.5.2 Java

- Changed naming scheme of GDX output scratch file to sequence number.

3.25.2.5.3 Python

- Fixed a bug in the constructor of all subclasses of GamsSymbol that occurred when the explanatory text was omitted.
- New property GamsSymbol.domains_as_strings: Domains of Symbol, each element is a string. If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.

3.25.2.6 Expert Level APIs

3.25.2.6.1 GMO

- Fixes to gmoGetRowJacInfoOne and gmoGetColJacInfoOne: In case of an empty row/column, now return -1 in colidx/rowidx if index base is 0.

3.25.2.7 Model Libraries

3.25.2.7.1 GAMS Model Library

- dyncge.gms: A Recursive-Dynamic Standard CGE Model (410)
272 Release Notes

3.25.3 24.4.3 Maintenance release (April 02, 2015)

3.25.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Guillaume Erbs and Emiliano Traversi.

3.25.3.2 Solvers

3.25.3.2.1 Examiner2

- Fixed incapability to continue on a trace file that already has trace records in it.

3.25.3.2.2 LocalSolver

- Corrected computation of values for free variables that appeared (linearly) in one equation only.

3.25.3.3 Tools

3.25.3.3.1 GDXXRW

- In 24.4.2 we quietly introduced a new way to determine the content of a sheet. This resulted in a bug for empty sheets and sheets that have been saved with an active filter. This has been fixed.

3.25.4 24.4.4 Maintenance release (May 12, 2015)

3.25.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Thomas Counsell, Michael Ferris, Jean Mercenier, and Loïc Ventre.

3.25.4.2 Solvers

3.25.4.2.1 Gurobi

- New libraries 6.0.4.
- The new libraries do not work on Windows XP anymore. As the library for COIN-OR solvers, SCIP, and SoPlex link to the Gurobi library, these solvers also do not work on Windows XP anymore.

3.25.4.2.2 Ipopt

- New libraries for Linux and Mac OS X.
  - Fixed an issue in the MA97 interface that lead to convergence problems.
3.25.4.2.3 Minos

- Consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 500 superbasics, limit the Hessian dimension to 500 even though the superbasic limit is larger.

3.25.4.2.4 Mosek

- New libraries 7.1.0.30.

3.25.4.2.5 Xpress

- New libraries Optimizer 27.01.08 (was 27.01.02 before).

3.25.4.3 Tools

3.25.4.3.1 GDXDUMP

- Fixed a problem when writing a scalar or scalar variable/equation in format CSV.

3.25.4.4 Object Oriented APIs

- Fixed a potential problem with GAMSModelinstance and certain OS culture settings.

3.25.4.5 Model Libraries

3.25.4.5.1 GAMS EMP Library

- transecs: Fixed formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.

3.25.5 24.4.5 Maintenance release (May 26, 2015)

3.25.5.1 Solvers

3.25.5.1.1 COIN-OR solvers, SCIP, SoPlex

- On Windows 32bit, OsiGurobi was split off into a separate library, so that other COIN-OR solvers (e.g., Bonmin, Cbc, Couenne, Ipopt) and SCIP and SoPlex do not require the Gurobi 6.0.4 library anymore. Thus, for this release, only Gurobi and OsiGurobi do not run on Windows XP anymore (see 24.4.4 notes on Gurobi).
3.25.6 24.4.6 Minor release (June 26, 2015)

3.25.6.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Rich Roberts, and Jens Schulz.

3.25.6.2 Solvers

3.25.6.2.1 CONOPT

- New libraries 3.16F.
- New option LMUSD for handling definitional equations.
- New option RVFILL for memory allocation when memory is tight.
- Changed a sorting procedure in the preprocessor. Models with some very dense rows and many pre-triangular variables can experience a significant speedup for the preprocessor.

3.25.6.2.2 CPLEX

- New libraries 12.6.2.0.
- Option changes:
  - The popular option value -1 for BarCrossAlg to turn off the crossover after a run with barrier has been deprecated. The new way to turn off crossover is to set the new option SolutionType to 2.
  - The option SolutionTarget has been renamed to OptimalityTarget.
  - The option CutsFactor has a new default (-1).
- New options:
  - BQPcuts: Boolean Quadric Polytope cuts for solving nonconvex QP or MIQP to global optimality.
  - CPUMask: Switch and mask to bind threads to processors. Binding threads to processors/cores helps to reduce variability in running time when using multiple threads.
  - SolutionType: Type of solution (basic or non basic) for an LP or QP. Set this option to 2 to prevent crossover after barrier.

3.25.6.2.3 MOSEK

- New libraries 7.1.0.31.

3.25.6.3 Tools

3.25.6.3.1 GAMSIDE

- Added .ref for reference file to the files open dialog.
- Left arrow in the first character position now moves to the end of the previous line.
3.26.3.2 GDXXRW

- Restore the old behavior when reading a set with Values=Strings; all elements will be included, not only the ones with a string.
  - Option Values=String and All are now deprecated and results in a warning; replaced with Dense.
  - New options Values=Dense or Sparse.

3.26.3.3 XLSTalk

- Allow for up to 9 parameters for macro call.

3.26 24.3 Distribution

3.26.1 24.3.1 Major release (July 31, 2014)

3.26.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Bond, Sebastian Bothor, Jared Erickson, Michael Ferris, Josef Kallrath, Bastian Niebel, Matt Preston, and Tom Rutherford.

3.26.1.2 Platforms

- The Linux 32bit and Solaris 32bit distributions have been dropped (as announced).
- Support for Windows XP may be dropped with some future GAMS version. As several of our solver vendors have already dropped support for Windows XP, some components of a GAMS system may cease to work under Windows XP in the future. If you notice that a 64bit GAMS system is not working on your Windows XP 64bit machine, please let us know.

3.26.1.3 GAMS System

3.26.1.3.1 GAMS

- Introduced new keyword Singleton Set: A Singleton Set in GAMS is a special Set that has at most one element (zero elements are allowed as well). Singleton Sets can be especially useful in assignment statements since they don’t need to be controlled by a controlling index nor an indexed operator, e.g.:

```gams
Set s / s1*s3 /;
Singleton Set single(s) / s2 /;
Parameter p(s);
Scalar x;

p(s) = ord(s);
```

```gams
x = p(single);
```
NOTE: Assigning membership to Singleton Sets is different than to usual sets. Since Singleton Sets can never have more than one element, any assignment to a Singleton Set first clears or empties the set, so no explicit clear is necessary. This is illustrated in the following example:

```
Set       i  Static Set    / a, b, c / 
          ii(i) Dynamic Set / b /;
Singleton Set   si(i) Dynamic Singleton Set / b /;
ii('c') = yes;
si('c') = yes;
Display ii, si;
```

Here is the output from the display statement in the listing file:

```
---- 8 SET ii Dynamic Set 
b, c
---- 8 SET si Dynamic Singleton Set 
c
```

More information can be found in the GAMS Users Guide.

- The behavior of assignments to Singleton Sets can be influenced by the new option strictSingleton [0/1]: This option affects the behavior of a membership assignment to a Singleton Set. With strictSingleton = 0 GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With strictSingleton = 1 (default), such an assignment raises an error. This option be set via an option statement and via a command line option.

- Similarly, data statements for Singleton Sets can be influenced by the new dollar control options $onStrictSingleton/$offStrictSingleton: With $offStrictSingleton GAMS does not complain about a data statement for a Singleton Set that has more than one element but takes the first one. With $onStrictSingleton (default), such an assignment raises an error.

- Introduced so-called obfuscated save files, which are save files where symbol and UEL names have been obfuscated. The new options saveobfuscate (so) and xsaveobfuscate (xso) can be used to generate uncompressed and compressed, respectively, obfuscated save files. Additionally, a new option restartNamed (rn) has been added which can be used to bring back the original names when restarting from an obfuscated save file. The intended use is the following.

- Compile (only) a GAMS model into a named and an obfuscated save file:
  
  `$call gams transport a=c s=0named saveobfuscate=0anon`

- Move the obfuscated save file to a non-secure machine and execute it there:
  
  `$echo * Empty > empty.gms`
  `$call gams empty r=0anon s=1anon`

- Bring the new (still obfuscated) save file with the results back to the safe machine and do a continued compilation with reporting and export. The continued compilation restarts from the obfuscated save file with all the results, etc., but gets a second save file with proper names through option restartNamed:

  `$call gams unload 'supply', supply.m; > unload.gms`

In this execution, everything is taken from the obfuscated restart file, except for the symbol and UEL names and the listing file title and subtitle, which are taken from the file specified via restartNamed.

- Allow to load the universe of labels from a GDX file into a set at run-time:
execute_load 'someFile', someSet=*

Note, that only labels known to the GAMS program will be loaded.

• Fixed a bug that caused wrong results when evaluating \texttt{prod} (or \texttt{smin/smax}) over an empty set inside \texttt{sum} like in the following example:

\begin{verbatim}
sets r / 1 /
s       / 1, 2 /
rs(r,s)  / 1.1 /
subrs(r,s) / 1.2 /;
scalar z1;
z1 = sum(rs, prod(subrs(rs), 2));
display z1;
\end{verbatim}

• Fixed a bug in calling error logging routine for extrinsic functions from within solvers.

3.26.1.3.2 Function libraries

• The extrinsic function library \texttt{cppcclib} has been expanded to include functions for the PDF and CDF of the trivariate normal distribution. Additionally, documentation for \texttt{cppcclib} has been added to Appendix J of the GAMS User's Guide and the relevant testlib models \texttt{cpplib00 thrucpplib05} have been added or updated.

3.26.1.4 Solvers

3.26.1.4.1 ANTIGONE, Bonmin, Cbc, Couenne, SCIP, Sulum

• Solvetrace files now include the GAMS input name (usually GAMS model name with \texttt{.gms} extension stripped) in the header line.

3.26.1.4.2 BARON

• New libraries 14.0.2
  – Significant advances in the handling of integer programs. In addition to several classes of integer cutting planes, calls to MIP solvers and hybrid LP/MIP/NLP relaxations for MINLPs have been added.
  – Complete rewrite of the interfaces to LP solvers.
  – Improvements in probing routines.
  – More robust checks for the validity of the solution of LP/NLP subsolvers.
  – Introduced interfaces to COIN-OR/OSI and CBC 2.8.9.
  – Increased numerical robustness for numerically challenging problems.
  – New range reduction techniques.
  – Systematic treatment of infeasible problems. With the \texttt{CompIIS} option, which by default is 0, the computation of an Irreducible Inconsistent Set (IIS) can be requested. Five different algorithms are available, with corresponding values of \texttt{CompIIS} equal to 1, ..., 5. Algorithm 1 is a fast heuristic, while algorithms 2 through 5 are more time consuming exact algorithms. If an IIS is found, it is reported in the GAMS listing file. BARON does not consider bounds on binary variables to be part of the IIS. For general integers, the option \texttt{IISint} can be be used to signal that general integers should be considered as potential members of the IIS, i.e, integrality constraints are questioned.
  • Fixed handling of BARON termination status when activating \texttt{DeltaTerm} option.
3.26.1.4.3 BONMIN

- Dropped MIP capability (use CBC instead).

3.26.1.4.4 CONOPT

- New libraries 3.16B.
  - The new option `Lsusdf` was added. If turned on (the default) CONOPT's preprocessor will look for definitional constraints which are constraints of the form $x = f(y)$ where the bounds on $x$ are wider than the range of the function $f$, given the bounds on $y$. CONOPT will search for and select a maximal set of independent definitional constraints. The dependent variable in a definitional constraints will be selected for the initial basis and CONOPT will try to use the definition to initialize $x$. There are two other options associated with this procedure: If `Lsusdf` is enabled (default), then only unique definitional constraints are selected. If it is disabled, then option `Lfusdf` specifies a limit (default 2) on the number of candidates a definitional constraint can have.
  - The option `Lsflsh` – flush the buffer, has now been implemented and is automatically turned on if there is a systems error.
  - The lower bound on option `Rtipvr` changed from 1.e-3 to 1.e-10.

3.26.1.4.5 Convert

- The AMPL writer supports special ordered sets, semicontinuous variables, and semiinteger variables now.
- Row names in `.lp` and `.mps` files now start with 'e' instead of 'c', so they match the names written by the `dict` option.

3.26.1.4.6 CPLEX

- New libraries 12.6.0.1.

3.26.1.4.7 CPLEX, SCIP, XPRESS

- Fixed handling of indicator constraints when specified with explicit labels.

3.26.1.4.8 DE

- New option `VaBigM` to control the Big M for a Value at Risk reformulation.
3.26.1.4.9 EMPSP

- New keywords \texttt{VaRUp} (=\texttt{VaR}) and \texttt{VaRLo}: These keywords can be used to optimize the Value at Risk for a certain confidence level. The syntax is similar to the one from \texttt{cVaRUp} (=\texttt{cVaR}) and \texttt{cVaRLo}:

\begin{verbatim}
VaR [rv var] scalar
\end{verbatim}

More information can be found here.

- Changed the order of the parameters for the triangular distribution from

\begin{verbatim}
randvar <name> triangular <low> <high> <mid>
\end{verbatim}

to

\begin{verbatim}
randvar <name> triangular <low> <mid> <high>
\end{verbatim}

3.26.1.4.10 Examiner2

- Fixed a bug in processing special ordered sets.

3.26.1.4.11 GUSS/Scenario Solver

- GUSS/Scenario solver can now be combined with the GAMS Grid Facility (see example \texttt{GUSSGRID} in the GAMS Model Library).

- New option \texttt{RestartType} to determines restart point for the scenarios:
  - 0: Restart from last solution (default)
  - 1: Restart from solution of base case
  - 2: Restart from input point

- New option \texttt{SolveEmpty} (default 0) to limit the number of empty scenarios (no scenario data) that are being solved. When the limit is reached, further empty scenarios will be skipped. Skipped scenarios will be reported to the log and listing file.

3.26.1.4.12 Ipopt

- New libraries.

3.26.1.4.13 Knitro

- As Ziena Optimization has dropped support for Knitro on Solaris (x86) a while ago, we plan to drop GAMS/Knitro on Solaris with the next release.

3.26.1.4.14 Lindo/LindoGlobal

- New libraries 8.0.550.

- LindoGlobal is no longer available for Sparc Solaris (as announced).
3.26.1.4.15 Mosek

- New libraries 7.0.0.121 (Linux, Mac OS X) and 7.0.0.123 (Windows).

3.26.1.4.16 OQNLP

- OQNLP is no longer available for Linux (as announced).

3.26.1.4.17 OS

- GAMS to OSiL conversion now creates more compact instances, especially for quadratic equations and long sums or products in general nonlinear expressions.

3.26.1.4.18 SCIP

- New libraries 3.1 #695c979.
  - Added new primal heuristics "random rounding", "proximity", and "dual value", new branching rule "cloud branching", and new node selectors "breadthfirst" and "uct".
  - Added support for strong branching with domain propagation in full strong and reliability pseudo cost branching.
  - Improved numerical stability (now taking the rank of cuts into account; more checks on LP solution; disabled scaling in feasibility check of nonlinear constraints).
  - Many improvements in presolving.
  - Strong branching LP solutions are now checked for feasibility.
  - Changed or removed parameters:
    - branching/relpscost/maxlookahead: default changed from 9 to 8.
    - branching/relpscost/maxreliable: default changed from 8 to 5.
    - constraints/bivariate/scaling: default changed from TRUE to 'o' (type changed to character).
    - constraints/quadratic/scaling: default changed from TRUE to 'o' (type changed to character).
    - constraints/soc/scaling: default changed from TRUE to 'o' (type changed to character).
    - constraints/varbound/maxlpcoef: default changed from 1E6 to 1E9.
    - heuristics/crossover/minnodes: default changed from 500 to 50.
    - heuristics/dins/minnodes: default changed from 500 to 50.
    - heuristics/feaspump/objfactor: default changed from 1 to 0.1.
    - heuristics/rens/minnodes: default changed from 500 to 50.
    - heuristics/rins/freq: default changed from -1 to 25.
    - heuristics/rins/freqofs: default changed from 5 to 0.
    - heuristics/rins/minfixingrate: default changed from 0 to 0.3.
    - heuristics/rins/minnodes: default changed from 500 to 50.
    - heuristics/shiftandpropagate/sortkey: default changed from 'u' to 'v'.
    - lp/checkfeas: replaced by new parameters lp/checkdualfeas and lp/checkprimfeas.
    - numerics/dualfeastol: default changed from 1E-6 to 1E-7.
    - presolving/dualfix/*: replaced by propagating/dualfix/*
    - propagating/pseudoobj/presoldelay: default changed from TRUE to FALSE.
    - propagating/pseudoobj/timingmask: default changed from 5 to 7.
    - propagating/redcost/timingmask: default changed from 2 to 6.
    - separating/cgmip/objweightsize: renamed to separating/cgmip/objweightsize and default changed from FALSE to TRUE.
    - separating/minefficacyroot: default changed from 0.01 to 0.001.
    - separating/closecuts/relintnormtype: removed
3.26.1.4.19 SoPlex

- New libraries 2.0.0.

3.26.1.4.20 XPRESS

- New libraries 26.01.08.

3.26.1.5 Tools

3.26.1.5.1 GDX2SQLITE

- GDX2SQLITE is a new tool to dump the complete contents of a GAMS GDX file into a SQLite database file.

3.26.1.5.2 GDXDUMP

- New option SymbolsAsSet to write the symbol table for a set as data.

3.26.1.5.3 GDXMRW

- New utilities irgdx and iwgdx for exchanging indexed GDX data with Matlab.

3.26.1.5.4 GDXRENAME

- New utility to rename the same unique elements in a GDX file using a mapping given by a second GDX file.

3.26.1.5.5 GDXXRW

- New option to allow the use of R1C1 notation to specify a cell or a range.
- Reading a set using the option Values=string now skips empty cells.

3.26.1.6 Expert Level APIs

3.26.1.6.1 GMO

- Removed previously deprecated function gmoDirtyExtractDefVar.
- Added a number of functions to access information from an EMP info file, to compute the optimality gap, to access extrinsic function libraries, and to get the name of the GAMS input file.
3.26.1.7 Object Oriented APIs

- New example transport14.
- New method `GAMSSymbolRecord.Key(int index)` (Java: `GAMSSymbolRecord.getKey`, Python: `GamsSymbolRecord.key(int index)`) to retrieve the key of `GAMSSymbolRecord` (Python: `GamsSymbolRecord`) for a given positional index.
- `GAMSSymbol.CopySymbol` now works for the universe of a `GAMSDatabase` (`GAMSDatabase.GetSet("*")`). When copying into the Universe, a merge will be performed.
- Real domains are now registered when exporting a `GAMSDatabase` to GDX (so far, only relaxed domains were registered).
- `GAMSJob.Run` (Python: `GamsJob.run`) now creates `OutDB` (Python: `out_db`) also if it raises a `GAMSExceptionExecution`.
- `GamsModelInstance.Solve` (Java/Python: `GamsModelInstance.solve`) now also works for solvers which require a subsolver, e.g., DICOPT.

3.26.1.7.1 .NET

- Fixed default value for `systemDirectory` argument in `GAMSWorkspace` constructor when using MONO: If no value is given, first the `PATH` and then the (DY)LD_LIBRARY_PATH is checked for a valid GAMS system directory.

3.26.1.7.2 Java

- It is no longer necessary to specify `-Djava.library.path` when running a program. If `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`. Otherwise, the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
- It is no longer necessary to set up environment variables (`PATH`, (DY)LD_LIBRARY_PATH) before running a program to find the GAMS system directory, as it can now be specified during run time.
- In the `GAMSWorkspace` default constructor, the default setting for finding the GAMS system directory from environment variables in the following order (depends on the target platform) are applied:
  - Windows: first from `PATH` environment variable. If not found, then from the Windows registry `gams.location`,
  - Mac OS X: first from `PATH` environment variable. If not found, then from `DYLD_LIBRARY_PATH`,
  - other Unix: from `PATH` environment variable. If not found, then from `LD_LIBRARY_PATH`.
- In the non-default `GAMSWorkspace` constructor, the following rules are applied:
  - In case a user specifies a system directory, the API will verify the directory and will not search for GAMS system directory from an environment variable, even when the directory is invalid.
  - In case the specified system directory is `null` or a user does not specify a system directory, the API will apply the default setting from above.
- A memory leak in `GAMSDatabaseIterator` has been closed.
- Issues when using a non-standard locale (`LANG` environment variable) have been fixed.
- All deprecated classes and methods since 24.1 have been removed.
3.26.1.7.3 Python

- New behavior on determining a system directory automatically when a workspace is created.
  - Linux: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check LD_LIBRARY_PATH.
  - Mac OS X: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check DYLD_LIBRARY_PATH.
  - Windows: If no system directory is specified in the GamsWorkspace constructor, check the Windows registry.

3.26.1.8 Model Libraries

3.26.1.8.1 GAMS Data Library

- MakeQL: Moved querylibrary generator into trisource.zip, which comes with Test Library models trilib01, trilib02, trilib03, and trilib04. Removed MakeQL model.
- invert1.gms: Matrix inversion via R (89)
- invert2.gms: Matrix inversion via Matlab (90)

3.26.1.8.2 GAMS Model Library

- clad.gms: Computation of Fairs extramarital affairs model estimates (397)
- gussex1.gms: Simple GUSS example (398)
- gussrisk.gms: Simple investment example with varying weight for risk using GUSS (399)
- gussgrid.gms: Simple GUSS Grid example (400)
- circpack.gms: Pack circles in the smallest possible rectangle (401)
- tablelayout.gms: Configuring text layout in table cells to minimize table height (402)
- asyncjobs.gms: Execute asynchronously several GAMS jobs and collect the fastest (403)

3.26.1.8.3 GAMS Test Library

- trilib01: Reworked triclib.c source code. Rewrote querylibrary generator.
- single01.gms: Check handling of singleton sets (639)
- execerr1.gms: Test for execerr option (640)
- single02.gms: Check assignment to singleton sets (641)
- scensol4.gms: Basic GUSS Test (642)
- gdxmrw06.gms: run a battery of GDXMRW tests (643)
- single03.gms: Check singleton sets in put statement (644)
• unload11.gms : Check that GAMS does not crash when writing to non-existing folder (645)
• refact00.gms : Check that GAMS produces expected workfile with option sys14=1 (646)
• refact01.gms : Refactor suite test 1 (647)
• scenempty.gms : Empty scenario GUSS Test (648)
• obfusc01.gms : Test use if obfuscated workfile (649)
• load11.gms : Load UEL Table (650)
• gussskip.gms : Simple GUSS example with skipped scenario (651)
• refact02.gms : Refactor suite test 2 (653)
• cpplib03.gms : Test extrinsic functions in cppcclib (654)
• cpplib04.gms : Test extrinsic functions in cppcclib (655)
• cpplib05.gms : Test extrinsic functions in cppcclib (656)
• cpplib00.gms : Test build of CPP library (657)
• syschk2.gms : Test impact of sys10 setting (658)
• scensol5.gms : Test handling of scenario dictionary sets with more than 50 entries (659)
• call5.gms : Check that gams works with COMSPEC unset (660)
• idxoper1.gms : Test indexed operations (661)
• gdxsqlite1.gms : Test basic functionality of GDX2SQLITE tool (662)

3.26.2 24.3.2 Minor release (August 29, 2014)

3.26.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Alvaro Lorca Galvez, Scott McDonald, Tom Rutherford, Nick Sahinidis, and Uwe Schneider.

3.26.2.2 GAMS System

3.26.2.2.1 GAMS

• Allow empty scalar data statements:

    $onempty
    scalar xyz / /;

3.26.2.2.2 Extrinsic Function libraries

• Pass on parameter file name via LibInit call.
3.26.2.3 Solvers

3.26.2.3.1 BARON

- New libraries 14.0.3.

3.26.2.3.2 CBC

- New libraries.

3.26.2.3.3 CONOPT

- New libraries 3.16C.
  - Added checks for internal intermediate results being NaN (Not A Number). If this happens, CONOPT will stop and return a message about it. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible."

3.26.2.3.4 CONVERT

- Fixed bug in interval evaluations for functions and gradients.
- The interval evaluations are now triggered by their own option (e.g. `intervalEval=yes`) instead of being part of every Jacobian or Hessian dump.

3.26.2.3.5 Cplex

- Some Cplex tuning parameters had no or the wrong effect. This has been fixed.
- The BCH usercutcall was not called in case of new constrains only for true cuts (see e.g. GAMS Model library model bchtsp). This has been fixed.

3.26.2.3.6 Examiner2

- Fixed error in processing options file.
- Fixed error in trace file generation.

3.26.2.3.7 Ipopt

- New libraries.

3.26.2.3.8 Mosek

- New libraries 7.0.0.126.
  - Fixed an issue with using the Mosek extended license for MIP solving on Mac OS X.
3.26.2.3.9 Osi links

- Fixed writing MPS files with row and column names.

3.26.2.3.10 SCIP

- New libraries 3.1 #322574a
  - Changed default of option heuristics/proximity/minimprove from 0.25 to 0.02.
  - Changed default of option heuristics/proximity/usefinallp from TRUE to FALSE.

3.26.2.3.11 XPRESS

- New libraries 26.01.14 for Linux and Windows.

3.26.2.4 Tools

3.26.2.4.1 GDXDUMP

- When generating $GDXIN filename, the filename now includes the full path of the GDX input file.

3.26.2.4.2 GDXXRW

- In the previous release, we changed the behavior that when reading with Values=String, the empty cells no longer created an entry. Because reading with this option was the default for the cases with RDim=0 or CDim=0, we changed this default to Values=NoData in order not to break existing code.

3.26.2.4.3 Shellexecute

- Fixed error causing problems during parameter processing.

3.26.2.5 Expert Level APIs

- Delphi: Distinguish between 32 bit and 64 bit compiler.
3.26.2.6 Object Oriented APIs

- Make more GAMS options available through the GAMSOptions class:
  - GAMSOptions.AppendExpand: Expand file append option
  - GAMSOptions.AppendOut: Output file append option
  - GAMSOptions.DumpOpt: Writes preprocessed input to the file input.dmp
  - GAMSOptions.DumpParms: GAMS parameter logging
  - GAMSOptions.ErrMsg: Placing of compilation error messages
  - GAMSOptions.Expand: Expanded (include) input file name
  - GAMSOptions.FErr: Alternative error message file
  - GAMSOptions.JobTrace: Job trace string to be written to the trace file at the end of a Gams job
  - GAMSOptions.LimCol: Maximum number of columns listed in one variable block
  - GAMSOptions.LimRow: Maximum number of rows listed in one equation block
  - GAMSOptions.LogLine: Amount of line tracing to the log file
  - GAMSOptions.On115: Generate errors for unknown unique element in an equation
  - GAMSOptions.Output: Output file
  - GAMSOptions.PageContr: Output file page control option
  - GAMSOptions.PageSize: Output file page size (=0 no paging)
  - GAMSOptions.PageWidth: Output file page width
  - GAMSOptions.Reference: Symbol reference file
  - GAMSOptions.ScriptExit: Program or script to be executed at the end of a GAMS run
  - GAMSOptions.Suppress: Compiler listing option
  - GAMSOptions.Symbol: Symbol table file
  - GAMSOptions.TraceLevel: Solvestate threshold used in conjunction with a=GT

3.26.2.7 Model Libraries

3.26.2.7.1 GAMS Data Library

- Fixed some errors in some Matlab examples.

3.26.3 24.3.3 Minor release (September 19, 2014)

3.26.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Henrik Dahl.

3.26.3.2 Solvers

3.26.3.2.1 ANTIGONE, Bonmin, CBC, Couenne, Ipopt, SCIP

- Fixed issue loading MKL libraries on Linux machines with AVX2 instruction set.
3.26.3.3  Tools

3.26.3.3.1  ASK, GDXVIEWER, SQL2GMS, XLS2GMS

- Fixed problem with character set used for error messages.

3.26.3.3.2  GDXXRW

- Fixed error when reading special values from Excel.

3.26.3.3.3  MDB2GMS, SQL2GMS

- Fixed problem when using comma as decimal separator.

3.26.3.4  Object Oriented APIs

3.26.3.4.1  Java

- Fixed a location of listing file when creating a job from (full-path) file without giving a job name.
3.27 24.2 Distribution

3.27.1 24.2.1 Major release (December 09, 2013)

3.27.2 24.2.2 Maintenance release (March 04, 2014)

3.27.3 24.2.3 Maintenance release (May 22, 2014)

3.28 24.1 Distribution

3.28.1 24.1.1 Major release (May 30, 2013)

3.28.2 24.1.2 Maintenance release (June 16, 2013)

3.28.3 24.1.3 Maintenance release (July 26, 2013)

3.29 24.0 Distribution

3.29.1 24.0.1 Major release (December 24, 2012)

3.29.2 24.0.2 Maintenance release (February 14, 2013)

3.30 23.9 Distribution

3.30.1 23.9.1 Major release (July 04, 2012)

3.30.2 23.9.2 Maintenance release (August 29, 2012)

3.30.3 23.9.3 Maintenance release (September 26, 2012)

3.30.4 23.9.4 Maintenance release (October 20, 2012)

3.30.5 23.9.5 Maintenance release (November 09, 2012)

3.31 23.8 Distribution

3.31.1 23.8.1 Major release (March 17, 2012)

3.31.2 23.8.2 Maintenance release (April 05, 2012)

- Bugfix for models with $=X=$ rows and MCP (i.e. MPSGE)
- GloMIQO resets $\text{OptCR}=0$ properly to $1e-9$ and accepts now Ctrl-C on Windows
- Knitro handles MINLP failures better
- XPRESS handles $=E=$ quadratic constraints better now
- Changed position of Library Form in GAMSIDE (avoid hang in corner to corner screen setup)
- Model Trnsxc11 added to GAMS Data Library
- Model ps5_a_sp added to GAMS Model Library
3.32 23.7 Distribution

3.32.1 23.7.1 Major release (July 14, 2011)

3.32.2 23.7.2 Maintenance release (July 22, 2011)

- Alphaecp: Alphaecp v2.04.02
- Conopt3: Fixed a system error 2003
- Dicopt: Fixed a bug for accidentally determined crossovers
- Inclib: updated put_reorderit.gms from Bruce McCarl
- Cbc: Disabled preprocessing in case of semicont/semiint variables, it seems buggy
- Cplex: Does no longer require finite bounds on semicont variables
- Gamsinst: Fix problem with blanks in sysdir and IDE error when not having a license

3.32.3 23.7.3 Maintenance release (August 23, 2011)

- alphaecp: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- apifiles:
  - get rid of warnings in cc interface,
  - fix problem with compilation of fat binaries on Darwin in cc interface
  - fix error in gdxSymbolGetDomain in Python interface
  - fix errors with constant definitions in C# interface
  - add constant definitions and additional constructor (which gets a handle) in Java interfaces
- conopt3: Version 3.15A maintenance release
- cplex:
  - fix for interrupt (Ctrl-C) when running with solvelink=5
  - fix memory leak
- dicopt:
  - fix reporting of objest in case of a crossover
  - fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- gamscmex: fix bug related to the internal organization of non-linear code, affects two-argument functions in rare cases
- gamside:
  - close chart files too when changing system directory
  - GAMS project file name independent of casing
- gdxdump: suppress on/off empty when writing single symbol
- gdxmrw: was missing on 64bit Linux
- grid: fix potential problem with scaling
• kestrel: added Xpress support
• minos fix: for interrupt (Ctrl-C)
• sbb: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
• snopt fix: for interrupt (Ctrl-C)
• xpress: update global search to use callbacks called during root node processing and improve scheme to stop XPRESS when gap is achieved or on user interrupt

3.33 23.6 Distribution

3.33.1 23.6.2 Major release (December 13, 2010)

3.33.2 23.6.3 Maintenance release (February 15, 2011)

• Cmex: Fix for variable level projection
• Bonmin: Several fixes for outer-approximation based algorithms
• Cplex: New version 12.2.0.2
• Gurobi: New version 4.0.1
• Knitro: Documentation update
• Scip: New version 2.0.1
• Xpress: Fix for crash on 64 bit Solaris
• McCarl Guide: Updates for 23.6
• Gdxxrw: Includes system error message when we cannot start Excel
• Gdxmrw: Minor bugfixes

3.33.3 23.6.4 Maintenance release (April 01, 2011)

• CMEX: Maximal nested includes were raised to 40
• Mosek: New Version6 Rev 105
• Lindoglobal: New Version 6.1.1.553
• Conopt: New Version 3.14W
• Xpress: Memory leak fix
• Coin: Solver fixes
• JAMS: Updates for disjunctions
• GDXXRW: Fixes
• GDXMRW: Updates
• McCarl and GDXUtils: Documentation updates
• Model Libraries: Updates
3.33.4 23.6.5 Maintenance release (April 08, 2011)

- CMEX: Bug fix related to situations with very large number of labels in connection with a restart with continued compilation

3.34 23.5 Distribution

3.34.1 23.5.1 Major release (July 05, 2010)

3.34.2 23.5.2 Maintenance release (August 18, 2010)

- AlphaECP: Fix for crash running under IDE
- API: Updated Python and Java APIs and examples
- Baron: Fixed issue with output formatting under Linux
- Baron: Memory leak when using BARON with external NLP solver
- Gdxmerge: Do not use path when checking for merged.gdx
- Gurobi: Fixed typo for option displayinterval
- McCarl Guide: Updates for 23.5
- Mosek 6 rev 85 libraries
- MPS2GMS: Removed mps2gms debug message
- Snopt: Fix for wrong return code (local,global)
- Xlsdump utility added

3.35 23.4 Distribution

3.35.1 23.4.1 Major release (May 21, 2010)

3.35.2 23.4.3 Maintenance release (May 24, 2010)

3.36 23.3 Distribution

3.36.1 23.3.1 Major release (November 01, 2009)

3.36.2 23.3.2 Maintenance release (November 18, 2009)

- Gurobi 2.0.1 library with bug fixes
- MOSEK 6 rev 53 library with bug fixes
- Updated CoinOS library with bug fixes
- Solver optfile bug fix in GAMSCHK
- Synchronized tolproj (1e-8) option for solvelink=3
### 3.37.3 23.3.3 Maintenance release (December 17, 2009)
- MOSEK 6 rev 55 library with bug fixes
- 32bit HAR utilities available in 64bit Windows System
- Slow solution reporting in GAMS/Gurobi and other solvers fixed

### 3.37 23.2 Distribution

### 3.37.1 23.2.1 Minor release (August 14, 2009)

### 3.38 23.1 Distribution

### 3.38.1 23.1.1 Major release (July 13, 2009)

### 3.38.2 23.1.2 Maintenance release (July 23, 2009)
- MPSGE: MPSGE ignored the \texttt{m.workspace} and \texttt{m.workfactor} settings
- GAMSCHK: Fixed a problem with the layout of the reports in some models

### 3.39 23.0 Distribution

### 3.39.1 23.0.2 Major release (February 14, 2009)

### 3.40 22.9 Distribution

### 3.40.1 22.9.2 Major release (December 01, 2008)

### 3.41 22.8 Distribution

### 3.41.1 22.8.1 Major release (August 01, 2008)

### 3.42 22.7 Distribution

### 3.42.1 22.7.1 Major release (May 01, 2008)

### 3.42.2 22.7.2 Maintenance release (May 13, 2008)
- DICOPT/ALPHAEC/LOGMIP/LOGBOA: Fixed Cplex scaling bug
- GAMS/CPLEX: Small cosmetic Cplex bug fix. We got "CPLEX Error 3003: Not a mixed-integer problem." In case we cannot solve the fixed problem. This was due to some query calls about nodes and iteration.
- GAMSIDE: Fixed a bug that could cause an out of memory error when moving a row or column to the plane index in the gdx data viewer
- GAMS/DEA: Avoid writing zeros to GDX files
- GAMS/LGOLIB: Fixed a memory leak
- Minor documentation updates
- Note: AIX and Mac OS X PPC were not updated.
3.43 22.6 Major release (December 24, 2007)
3.44 22.5 Major release (June 01, 2007)
3.45 22.4 Major release (February 12, 2007)
3.46 22.3 Major release (November 27, 2006)
3.47 22.2 Minor release (April 21, 2006)
3.48 22.1 Major release (March 15, 2006)
3.49 22.0 Major release (August 01, 2005)
3.50 21.7 Major release (April 01, 2005)
3.51 21.6 Minor release (January 26, 2005)
3.52 21.5 Minor release (November 11, 2004)
3.53 21.4 Major release (September 06, 2004)
3.54 21.3 Major release (January 19, 2004)
3.55 21.2 Maintenance release (September 03, 2003)
3.56 21.1 Maintenance release (June 02, 2003)
3.57 21.0 Major release (May 15, 2003)
3.58 20.7 Maintenance release (June 14, 2002)
3.59 20.6 Major release (May 25, 2002)
3.60 20.5 Maintenance release (January 28, 2002)
3.61 20.4 Maintenance release (January 21, 2002)
3.62 20.3 Major release (December 24, 2001)
3.63 20.2 Maintenance release (November 22, 2001)
3.64 20.1 Major release (October 31, 2001)
Chapter 4

User's Guide

This documentation guides GAMS users through several topics in the GAMS system.

4.1 Installation and Licensing

This part leads step by step through the installation process on three main platforms and describes the GAMS licensing system:

- Supported Platforms
- Installation Notes for Mac OS X
- Installation Notes for Unix
- Installation Notes for Windows
- Standard Locations
- Licensing

4.2 Tutorials and Examples

This part describes step by step through several selected tutorials and a small number of examples.

- A GAMS Tutorial by Richard E. Rosenthal
- Quick Start Tutorial
- Good Coding Practices
- Fixing Compilation Errors
- Finding and Fixing Execution Errors and Performance Problems
- Comparative Analyses with GAMS
- Good NLP Formulations
- Data Exchange with Other Applications
4.3 GAMS Language and Environment

This part introduces the components of the GAMS language in an ordered way, interspersed with detailed examples that are often drawn from the model library.

- Introduction - an introduction to the GAMS User’s Guide.
- GAMS Programs - The structure of the GAMS language and its components
- Set Definition - The declaration and initialization of sets, subsets, and domain checking.
- Dynamic Sets - The membership assignment, the usage of dollar controls, and set operations.
- Sets as Sequences: Ordered Sets - Special features used to deal with a set as if it were a sequence.
- Data Manipulations with Parameters - The declaration and assignment of GAMS parameters.
- Data Entry: Parameters, Scalars and Tables - Three basic forms of GAMS data types: Parameters, Scalars and Tables.
- Variables - The declaration and attributes of GAMS variables.
- Equations - The definition and declaration of GAMS equations.
- Model and Solve Statements - The specification of a GAMS model and how to solve it.
- Conditional Expressions, Assignments and Equations - The conditional assignments, expressions and equations in GAMS.
- The Display Statement - The syntax, control, and label order in display.
- Programming Flow Control Features - The GAMS programming flow control features: loop, if-elseif, while, and for statements.
- The Option Statement - The list and detailed description of options.
- System Attributes - The system attributes
- The Grid and Multi-Threading Solve Facility - The basic concepts and Grid Features.
- Special Features for Mathematical Programs - Special features in GAMS that do not translate across solvers, or are specific to certain model types.
The following discusses the execution of GAMS, the use of special features, and other miscellaneous topics.

- **GAMS Output** - The control of GAMS compilation output, execution output, output produced by a solve statement, and error reporting.
- **GAMS Log**
- **The GAMS Call and Command Line Parameters** - The list and detailed description of GAMS command line parameters.
- **Dollar Control Options** - The list and detailed description of dollar control options.
- **The Put Writing Facility** - The put writing facility of the GAMS language.
- **Solver Usage** - Controlling solvers.
- **The Save and Restart Feature** - The GAMS save and restart feature and the work file.
- **Embedded Code Facility** - The Embedded Code Facility (e.g. how to embed Python code into GAMS).
- **GAMS Connect** - The GAMS Connect Framework.
- **Extrinsic Functions** - The extrinsic function library and comparison with external equations.
- **External Equations** - A facility for connecting code written in different programming languages to equations and variables in a GAMS model.
- **GAMS Return Codes** - The structure of error codes, the return codes of the GAMS compiler and execution system, and the driver return codes.
- **GAMS Data eXchange (GDX)** - GAMS Data eXchange (GDX) facilities and utilities for Binary Data Exchange.
- **Extended Mathematical Programming (EMP)** - Extended Mathematical Programming (EMP).
- **Accessing Model Libraries** - Introduction of GAMS Model Library.
- **Mathematical Programming System for General Equilibrium analysis (MPSGE)** - A mathematical programming system for general equilibrium analysis which operates as a subsystem within GAMS.
  - Introduction to MPSGE
  - MPSGE Models in GAMS
  - Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE
  - Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae
  - A Library of Small Examples for Self-Study
  - Linking Implan Social Accounts to MPSGE
  - The MPSGE guide is also available as PDF

### 4.4 Glossary

An alphabetical list of GAMS terms is available in the **Glossary**.
4.5 Supported Platforms

GAMS divides the set of supported platforms into so-called core platforms and peripheral platforms to recognize and better describe how the two sets of platforms have evolved. The user communities for core platforms are large, active, and well-identified, but not so for the peripheral platforms. For the core platforms, GAMS will continue to make each new release of GAMS directly available for download, while the peripheral platforms will be available by request only. Finally, changes in the availability of core platforms will be announced well in advance of the event, while changes to support for peripheral platforms may occur with little or no notice.

The core platforms are the following:
Note

Compared to most of the GAMS system, GAMS Studio has some additional system requirements which are detailed here.

There are currently no peripheral platforms.

While the GAMS execution system itself is available on all supported platforms, for certain solvers, tools, and APIs, different availabilities can apply. For details, see

- Supported Platforms for Solvers
- Supported Platforms for Tools
- Supported Platforms for High-level APIs
- Supported Platforms for Low-level APIs

### 4.6 Installation Notes for Mac OS X

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Furthermore, we recommend to use the PKG installer for Mac OS X because it includes the GAMS Studio and it integrates GAMS into Mac OS X, e.g. it is possible to open the GAMS Studio via the Launchpad.

Two installation procedures are available for GAMS on Mac OS X:

- Installation using the PKG installer (GAMS40.1.0.pkg)
- Installation using the self-extracting archive (osx_x64.sfx.exe)

### 4.6.1 Installation using the PKG installer (GAMS40.1.0.pkg)

1. Obtain the GAMS PKG file, which is available from [http://www.gams.com/download](http://www.gams.com/download).
2. Double click the package and follow the instructions. By clicking on Customise in the tab Installation Type you can decide to add GAMS to the PATH as well as to reject the installation of GAMS Studio. Note that adding GAMS to the PATH is done by modifying your shell profile file (~/.zprofile, ~/.bash_profile etc.). A backup of your old profile is saved (~/.zprofile.gamsbackup, ~/.bash_profile.gamsbackup etc.).
3. In order to test the GAMS installation with GAMS Studio open the Launchpad and click the GAMS Studio 40 icon to open the application. Alternatively, go to Applications and open the GAMS Studio 40 application.
4. Install your license via the corresponding GAMS Studio dialog. The license file is nowadays sent via email, with instructions. You can also request a demo license from http://www.gams.com/download.

Optionally, you can create the license file 'gamslice.txt' in a directory that GAMS searches to find a license. GAMS searches a couple of system wide and user specific standard locations for a license file.

5. Open the Model Library Explorer and open the TRANSPORT model (sequence number 001). Run the model and check the contents of the process log, which should be similar to the Terminal output listed in the next bullet point.

6. In order to test the GAMS installation without using GAMS Studio open a Terminal window. Execute the following commands to see if everything works as expected:

```
gamslib trnsport
gams trnsport
```

The output should be similar to this:

```
--- Job trnsport Start 06/26/14 11:24:56 24.3.1 r46409 DEX-DEG Mac x86_64/Darwin
GAMS 24.3.1  Copyright (C) 1987-2014 GAMS Development. All rights reserved
Licensee: ...
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.114
IBM ILOG CPLEX 24.3.1 ... DEG Mac x86_64/Darwin
Cplex 12.6.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.02 sec. (0.00 ticks)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Dual Objective</th>
<th>In Variable</th>
<th>Out Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73.125000</td>
<td>x(seattle.new-york)</td>
<td>demand(new-york) slack</td>
</tr>
<tr>
<td>2</td>
<td>119.025000</td>
<td>x(seattle.chicago)</td>
<td>demand(chicago) slack</td>
</tr>
<tr>
<td>3</td>
<td>153.675000</td>
<td>x(san-diego.topeka)</td>
<td>demand(topeka) slack</td>
</tr>
<tr>
<td>4</td>
<td>153.675000</td>
<td>x(san-diego.new-york)</td>
<td>supply(seattle) slack</td>
</tr>
</tbody>
</table>

LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective : 153.675000
--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
 *** Status: Normal completion
--- Job trnsport.gms Stop 06/26/14 11:24:57 elapsed 0:00:00.487
4.6.2 Uninstall PKG installation

To uninstall a GAMS installation that was installed using the PKG installer, run the following command from the terminal: `sudo bash /Library/Frameworks/GAMS.framework/Versions/40/Resources/uninstall.sh`. Additionally, you can remove GAMS from the PATH in your shell profile file (e.g., `~/.zprofile`, `~/.bash_profile`) if GAMS was added to the PATH during installation.

4.6.3 Installation using the self-extracting archive (osx_x64_64_sfx.exe)

1. Obtain the GAMS distribution file, which is available from [http://www.gams.com/download](http://www.gams.com/download), in one large self-extracting zip archive with a `sfx.exe` file extension, e.g., `osx_x64_64_sfx.exe`. Check that it has the execute permission set. If you are not sure how to do this, just type in the command `chmod 755 osx_x64_64_sfx.exe`.

Attention

The common way to install GAMS on a Mac is the PKG installer. When one tries to run the sfx installer (e.g. for unattended installation) under macOS 10.15 (Catalina) or newer, the installer and several related files will be tagged with the `com.apple.quarantine` flag. There are different solutions to this problem:

- The flag can be removed by the following command:
  ```bash
  xattr -rd com.apple.quarantine osx_x64_64_sfx.exe
  ```
- Instead of downloading the sfx installer through the browser, use a command line tool such as curl (note that by downloading the software, you agree to the License Agreement). The download link may need to be adjusted, depending on the distribution that should be downloaded.
  ```bash
  curl https://d37drm4t2jghv5.cloudfront.net/distributions/29.1.0/macosx/osx_x64_64_sfx.exe
  ```

2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in `/Applications/GAMS`, the installer will create the GAMS system directory `/Applications/GAMS/gams24.3.osx_x64_64_sfx`. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.

3. Create the directory that should contain the GAMS system directory, for instance `/Applications/GAMS`. Change to this directory (`cd /Applications/GAMS`). Make sure `pwd` returns the name of this directory correctly.

4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

   ```bash
   mkdir /Applications/GAMS
   cd /Applications/GAMS
   ~/osx_x64_64_sfx.exe
   ```

5. Create the license file `gamslicetxt` in a directory GAMS searches to find a license. The license file is nowadays sent via email, with instructions. You can also request a demo license from [http://www.gams.com/download](http://www.gams.com/download).
Attention

Do not store the gamslice.txt in the GAMS system directory. This invalidates the code signature and cause Gatekeeper to reject the bundle!

GAMS searches a couple of system wide and user specific locations for a license file. For Mac OS X these locations include /Library/Application Support/GAMS and /Users/username/Library/Application Support/GAMS. The locations can vary due to different system configuration. One can get an ordered list of data directories GAMS searches for gamslice.txt by running the program ./gamsinst -listdirs from the GAMS system directory. Even though this list might contain locations in the system directory, e.g. /Applications/GAMS31.1 we strongly discourage to place gamslice.txt here.

6. Change to the GAMS system directory and run the program ./gamsinst. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:
   a. rerunning ./gamsinst and resetting the default values
   b. setting a command line default, e.g., gams transport lp=soplex
   c. an option statement in the GAMS model, e.g: option lp=soplex;

7. Add the GAMS system directory to your path (see below).

8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

   LP: transport (objective value: 153.675)
   NLP: chenery (objective value: 1058.9)
   MIP: bid (optimal solution: 15210109.512)
   MINLP: procsel (optimal solution: 1.9231)
   MCP: scarfmc (no objective function)
   MPSGE: scarfmg (no objective function)

9. If you move the GAMS system to another directory, remember to rerun ./gamsinst. It is also good practice to rerun ./gamsinst when you change your license file if this has changed the set of licensed solvers.

4.6.3.1 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (csh) and its variants you can modify your .cshrc file by adding the second of the two lines given below:

   set path = (/your/previous/path/setting )
   set path = ( $path /Applications/GAMS/gams24.3_osx_x64_64_sfx ) # new

2. Those of you using the Bourne (sh) or Korn (ksh) shells and their variants can modify their .profile file by adding the second of the three lines below:

   PATH=/your/previous/path/setting
   PATH=$PATH:/Applications/GAMS/gams24.3_osx_x64_64_sfx # new
   export PATH

   If the .profile file does not exist yet, it needs to be created. You should log out and log in again after you have made any changes to your path.
3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their .cshrc file:

```
alias gams /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
alias gamslib /Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
```

The correct Bourne or Korn shell syntax (either command line or .profile) is:

```
alias gams=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
alias gamslib=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
```

Again, you should log out and log in order to put the alias settings in .cshrc or .profile into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.:

```
/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams trnsport
```

### 4.6.3.2 Example

The following shows the log of a session, where a user downloads a GAMS 24.3.1 system and installs it under Applications/GAMS/gams24.3_osx_x64_64_sfx. It is assumed that a GAMS license file has been stored as /Users/doe/gamslice.txt.

```
doe@mac:/Users/doe$ curl -L -k -O \\nhttp://d37drm4t2jghv5.cloudfront.net/distributions/24.3.1/macosx/osx_x64_64_sfx.exe
% Total  % Received  % Xferd  Average Speed   Time   Time  Time     Current
     100  148M  100  148M    0     0  4374k --:--:-- --:--:--  5906k

doe@mac:/Users/doe$ chmod 755 osx_x64_64_sfx.exe

doe@mac:/Users/doe$ cd /Applications/GAMS

doe@mac:/Applications/GAMS$ ~/osx_x64_64_sfx.exe
    creating: gams24.3_osx_x64_64_sfx/
    inflating: gams24.3_osx_x64_64_sfx/sp2full.m
    inflating: gams24.3_osx_x64_64_sfx/optpathnlp.def
    inflating: gams24.3_osx_x64_64_sfx/MessageReceiverWindow.exe
    inflating: gams24.3_osx_x64_64_sfx/datalib
    inflating: gams24.3_osx_x64_64_sfx/empsyntax.txt
    inflating: gams24.3_osx_x64_64_sfx/optlindoglobal.html
    ...
    inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.cs
    inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.csproj
    inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/xp_example2.cs
    inflating: gams24.3_osx_x64_64_sfx/optdicopt.html

doe@mac:/Applications/GAMS$ cd gams24.3_osx_x64_64_sfx

doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$/gamsinst -license ~/gamslice.txt
```
LP (Linear Programming) models can be solved by:
1. CBC (demo or student license)
2. CONOPT (demo or student license)
3. CPLEX (demo or student license)
4. GUROBI (demo or student license)
5. IPOPT (demo or student license)
6. IPOPTH (demo or student license)

Installed defaults:

<table>
<thead>
<tr>
<th>Type</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>CPLEX</td>
</tr>
<tr>
<td>MIP</td>
<td>CPLEX</td>
</tr>
<tr>
<td>RMIP</td>
<td>CPLEX</td>
</tr>
<tr>
<td>NLP</td>
<td>CONOPT</td>
</tr>
<tr>
<td>MCP</td>
<td>PATH</td>
</tr>
<tr>
<td>MPEC</td>
<td>NLPEC</td>
</tr>
<tr>
<td>RMPEC</td>
<td>NLPEC</td>
</tr>
<tr>
<td>CNS</td>
<td>CONOPT</td>
</tr>
<tr>
<td>DNLP</td>
<td>CONOPT</td>
</tr>
<tr>
<td>RMINLP</td>
<td>CONOPT</td>
</tr>
<tr>
<td>MINLP</td>
<td>DICOPT</td>
</tr>
<tr>
<td>QCP</td>
<td>CONOPT</td>
</tr>
<tr>
<td>MIQCP</td>
<td>SBB</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>CONOPT</td>
</tr>
<tr>
<td>EMP</td>
<td>JAMS</td>
</tr>
</tbody>
</table>

We are now prepared to set read and execute permissions on the GAMS system files. If you are not sure which option to choose, we recommend option 3.
You can set read (and execute) permission for:
1. user only.
2. user and group.
3. user, group, and world.
Enter your choice now

3

The files "gams", "gamslib", etc., are now executable.
You can run these commands in a number of ways:
1. Call them using an absolute path (i.e. /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams).
2. Create your own aliases for them.
3. Add the GAMS system directory "/Applications/GAMS/gams24.3_osx_x64_64_sfx" to your path.
Method 3. is recommended.
4.7 Installation Notes for Unix

4.7.1 Installation

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Additionally, a video on how to install GAMS on Linux is available at https://www.youtube.com/watch?v=Mx_tYI3wyP4.

1. Obtain the GAMS distribution file, which is available from http://www.gams.com/latest, in one large self-extracting zip archive with a _sfx.exe file extension, e.g., linux_x64_64_sfx.exe on a Linux 64bit system. Check that it has the execute permission set. If you are not sure how to do this, just type in the command, e.g., chmod 755 linux_x64_64_sfx.exe.
2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in /opt/gams, the installer will create the GAMS system directory /opt/gams/gams24.3_linux_x64_64.sfx. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.

3. Create the directory that should contain the GAMS system directory, for instance /opt/gams. Change to this directory (cd /opt/gams). Make sure pwd returns the name of this directory correctly.

4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

```
mkdir /opt/gams
cd /opt/gams
~/linux_x64_64_sfx.exe
```

5. Create the license file gamslice.txt in a directory GAMS searches to find a license or use the corresponding GAMS Studio dialog. The license file is nowadays sent via email, with instructions. You can also request a demo license from http://www.gams.com/download. GAMS searches a couple of system wide and user specific standard locations for a license file.

6. Change to the GAMS system directory and run the program ./gamsinst. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:
   a. rerunning ./gamsinst and resetting the default values
   b. setting a command line default, e.g., gams trnsport lp=soplex
   c. an option statement in the GAMS model, e.g: option lp=soplex;

7. Add the GAMS system directory to your path (see below).

8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

```
LP: trnsport (objective value: 153.675)
NLP: chenery (objective value: 1058.9)
MIP: bid (optimal solution: 15210109.512)
MINLP: procsel (optimal solution: 1.9231)
MCP: scarfmcp (no objective function)
MPSGE: scarfmge (no objective function)
```

9. If you move the GAMS system to another directory, remember to rerun ./gamsinst. It is also good practice to rerun ./gamsinst when you change your license file if this has changed the set of licensed solvers.

### 4.7.2 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (csh) and its variants you can modify your .cshrc file by adding the line
4.8 Installation Notes for Windows

4.8.1 Installation

1. Run windows_x64_64.exe (Windows 64bit): This file is available from https://www.gams.com/latest. Please note that the installation may require administrative privileges on your machine.

You have two options to run the installer: In default or advanced mode. In the default mode, the installer will prompt you for the name of the directory in which to install GAMS. We call this directory the **GAMS directory**. You may accept the default choice or pick another directory. Please remember: if you want to install two different versions of GAMS, they must be in separate directories.

If you choose to use the advanced mode, the installer will also ask you for a name of a start menu folder, if GAMS should be installed for all users, if the **GAMS directory** should be added to the **PATH** environment variable and which desktop icons should be created.

The **GAMS License** page allows to select a GAMS license. The installer tries to find GAMS licenses in different locations and automatically selects a license to be used with the following precedence:

- Clipboard
- C:\Users\username\Documents\GAMS
- Previous GAMS installation found from the registry (User)
- Previous GAMS installation found from the registry (System)
In addition it is possible to browse for a license file (gamslice.txt) or to skip all license related actions (No License) and install one after the installation process. Note that the use of GAMS requires a license. A free demo license can be requested at https://www.gams.com/download. A license file is written to C:|Users\username\Documents\GAMS when installing for Current User (default). Installing for All Users or checking Write License to System Directory will write the license to the GAMS system directory instead.

For automating the installation of GAMS, it is possible to provide the command line parameters /SP=  /SILENT. This will install GAMS in a non-interactive manner using default settings. Note that depending on the security settings, the User Account Control asking for permission might still be active. The following command line parameters are available to further customize the non-interactive installation:

- /installAllUsers=yes|no (default:no): Can be used in order to install for All Users instead of Current User.
- /noLicense=yes|no (default:no): Can be used in order to prevent any GAMS license from being written.
- /noIcons can be used in order to check the Don't create a Start Menu folder checkbox on the Select Start Menu Folder page.
- /desktopIcons=yes|no (default:undef) can be used in order to turn on/off the creation of Desktop icons. The default (undef) is to create a desktop icon for GAMS Studio but not for GAMS IDE.
- /allUsers: Perform installation in administrative install mode.
- /currentUser: Perform installation in non administrative install mode. If specified, /installAllUsers=yes has no effect.

Attention

Under Windows 10 the Windows Installer package may not update system environment variables (see Microsoft Forum). To work around this issue, log off of Windows after the installation, and then log on again.

Note

The installer can be run unattended with the switches /SILENT or /VERYSILENT and might require some post installation task (like copying the license file to an appropriate location). This unattended mode also skips execution of some code in the installer that can cause failures on exotically configured systems.

2. During the installation of GAMS, the setup process helps in setting up the license to be used with GAMS. In case no license has been selected during this process it is required to set this up manually. The license file is nowadays sent via email, with instructions. You can also request a demo license from https://www.gams.com/download. If you don't have a license file yet, choose No license when asked for the GAMS license options. You can install a license later by either manually creating the license file gamslice.txt in a directory GAMS searches to find a license or use the corresponding GAMS Studio dialog.

GAMS searches a couple of system wide and user specific standard locations for a license file.

3. Choose the default solvers: Run GAMS Studio by double clicking studio.exe from the GAMS system subdirectory Studio. To view or edit the default solvers, open the GAMS Configuration Editor in Studio. You can also skip this and thus accept the existing defaults if you wish, but most users want to select new default solvers for each model type.

4. Run a few models to test the GAMS system: The Model Library Explorer allows you to pick some models from the GAMS model library easily. After selecting one, press F9 to run it, and view the solution. To test your installation, run the following models from the GAMS model library:

LP: transport (objective value: 153.675)
NLP: chenery (objective value: 1058.9)
4.8 Installation Notes for Windows

MIP: bid (optimal solution: 15210109.512)
MINLP: procsel (optimal solution: 1.9231)
MCP: scarfmcp (no objective function)
MPSGE: scarfmge (no objective function)

4.8.2 Visual C++ Redistributable Dependency

- Some solvers in the system as well as GAMS Studio have dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio or when solving a model, please run the installer vcredist_x64.exe for these libraries, which can be found in the GAMS installation folder in directory \[GAMS system\]\apifiles\C++\lib\vs2017.

4.8.3 Command Line Use of GAMS

Users wishing to use GAMS from the command line (aka the console mode) may want to perform the following steps. These steps are not necessary to run GAMS via Studio.

1. We recommend to add the GAMS directory to your environment path in order to avoid having to type in an absolute path name each time you run GAMS. Run the installer in advanced mode and mark the check-box **Add GAMS directory to PATH environment variable**.

2. Run the program **gamsinst**: gamsinst is a command line program used to configure GAMS. It prompts the user for default solvers to be used for each model type. If possible, choose solvers you have licensed, since unlicensed solvers will only run in demonstration mode. The solver defaults can be changed by:
   a. rerunning gamsinst and resetting the default values
   b. setting a command line default, e.g. **gams trnsport lp=soplex**
   c. by an option statement in the GAMS model, e.g: **option lp=soplex;**

   The system wide solver defaults are shared by the command line and GAMS Studio, so you can also choose to set these defaults using GAMS Studio.

4.8.4 Warning from Microsoft SmartScreen Filter

The SmartScreen is a feature that is used in several Microsoft products, including the Internet Explorer, Microsoft Edge, as well as in Windows 8 and newer versions of Windows. It is supposed to help to protect your computer against unwanted software. It uses a reputation-based system to rate files downloaded from the internet. That means that a file gets a better reputation when it gets download more often. As a result you might get a warning like this, especially for GAMS distributions that were recently released:

clicking on “More info” will display an option to continue the installation process.
4.9 Standard Locations

GAMS looks for specific files in some system-wide and user-specific locations which are independent of the GAMS installation. This allows the user to store these files in one place without the need to adjust anything when moving from one GAMS version to another. These folders are referred to as standard locations and are split into two groups: configuration directories and data directories. Here are some examples of files, which will be found in the data directories (or subfolders of these):

- The GAMS license file
- LibInclude files
- External Equation libraries
- Extrinsic Function libraries

The GAMS configuration file is expected in one of the configuration directories.

The exact location depends on the operating system used and user specific settings. Running gamsinst -listdirs on the command line shows the folders searched explicitly.

4.9.1 Standard Locations on Mac OS X

4.9.1.1 Configuration Directories

These folders get searched on Mac OS X (in the listed order):

- The GAMS system directory
- The subfolder GAMS in $HOME/Library/Preferences ($HOME usually contains the user's home directory)

Here is an example output of gamsinst -listdirs:

Config directories (searched for e.g. gamsconfig.yaml):
/Users/JaneDoe/GAMS32
/Users/JaneDoe/Library/Preferences/GAMS
4.9.1.2 Data Directories

These folders get searched on Mac OS X (in the listed order):

- The subfolder GAMS in $HOME/Library/Application Support ($HOME usually contains the user's home directory)
- The subfolder GAMS in /Library/Application Support
- The subfolder Resources next to the GAMS system directory
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

```
Data directories (searched for e.g. gamslice.txt):
/Users/JaneDoe/Library/Application Support/GAMS
/Library/Application Support/GAMS
/Users/JaneDoe/Resources
/Users/JaneDoe/GAMS32
```

4.9.2 Standard Locations on Unix

4.9.2.1 Configuration Directories

On Unix the environment variables XDG_CONFIG_HOME and XDG_CONFIG_DIRS (see XDG Base Directory Specification) influence the search directories. These folders get searched on Unix (in the listed order):

- The GAMS system directory
- The subfolder GAMS of folders from $XDG_CONFIG_DIRS or the subfolder GAMS in /etc/xdg
- The folder $XDG_CONFIG_HOME or the subfolder GAMS in $HOME/.config ($HOME usually contains the user's home directory)

Here is an example output of `gamsinst -listdirs`:

```
Config directories (searched for e.g. gamsconfig.yaml):
/home/JohnDoe/GAMS32
/etc/xdg/GAMS
/home/JohnDoe/.config/GAMS
```

4.9.2.2 Data Directories

On Unix the environment variables XDG_DATA_HOME and XDG_DATA_DIRS (see XDG Base Directory Specification) influences the search directories. These folders get searched on Unix (in the listed order):

- The folder $XDG_DATA_HOME or the subfolder GAMS in $HOME/.local/share ($HOME usually contains the user's home directory)
- The subfolder GAMS of folders from $XDG_DATA_DIRS or the subfolder GAMS in /usr/local/share and the subfolder GAMS in /usr/share
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

```
Data directories (searched for e.g. gamslice.txt):
/home/JohnDoe/.local/share/GAMS
/usr/local/share/GAMS
/usr/share/GAMS
/home/JohnDoe/GAMS32
```
4.9.3 Standard Locations on Windows

4.9.3.1 Configuration Directories

These folders get searched on Windows (in the listed order):

- The GAMS system directory
- The subfolder GAMS in C:\ProgramData
- The subfolder GAMS in AppData\Local in the User's directory
- The subfolder GAMS in Documents in the User's directory

Here is an example output of `gamsinst -listdirs`:

Config directories (searched for e.g. gamsconfig.yaml):

```
C:\GAMS\32
C:\ProgramData\GAMS
C:\(Users\JohnDoe\AppData\Local\GAMS
C:\(Users\JohnDoe\Documents\GAMS
```

4.9.3.2 Data Directories

These folders get searched on Windows (in the listed order):

- The subfolder GAMS in Documents in the User's directory
- The subfolder GAMS in AppData\Roaming in the User's directory
- The subfolder GAMS in C:\ProgramData
- The subfolder data in the GAMS system directory
- The subfolder data\GAMS in the GAMS system directory
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

Data directories (searched for e.g. gamslice.txt):

```
C:\(Users\JohnDoe\Documents\GAMS
C:\(Users\JohnDoe\AppData\Roaming\GAMS
C:\ProgramData\GAMS
C:\GAMS\32\data
C:\GAMS\32\data\GAMS
C:\GAMS\32
```

Note: When running as a service, the "User's directory" mentioned above is not C:\Users\UserName but resolves to something like C:\Windows\system32\config\systemprofile. So the above output could look like this:

```
Data directories (searched for e.g. gamslice.txt):

C:\Windows\system32\config\systemprofile\Documents\GAMS
C:\Windows\system32\config\systemprofile\AppData\Roaming\GAMS
C:\ProgramData\GAMS
C:\GAMS\32\data
C:\GAMS\32\data\GAMS
C:\GAMS\32
```
4.10 Licensing

4.10.1 General Information

The GAMS system and all connected solvers can be downloaded from the GAMS website at https://www.gams.com/download. The use of the software is governed by the GAMS End User License Agreement, which is available on our website and can also be found in the GAMS System directory (eula.pdf). Without a valid GAMS license the system will not run and produces the following output:

*** No license specified and no gamslice.txt found in standard locations.

You can request a free demo license at https://www.gams.com/download. The demo license lets you generate and solve linear models (LP, RMIP, and MIP) that do not exceed 2000 variables and 2000 constraints. For all other model types the model cannot be larger than 1000 variables and 1000 constraints. Some solvers might enforce additional limits for the demo license. The demo license of MIRO connector allows up to 10 indexed input and output symbols.

GAMS will terminate with an error message, if it hits one of the limits above:

*** Status: Terminated due to a licensing error
*** Inspect listing file for more information

The demo license is time-limited for 12 months and on expiration you need to request a new license. Moreover, a valid free demo license requires a recent GAMS version (not older than 18 months).

4.10.2 GAMS Community License

Users can request a free community license from community@gams.com. The community license lets you generate and solve linear models (LP, MIP, and RMIP) that do not exceed 5000 variables and 5000 constraints. For all other model types the model cannot be larger than 2500 variables and 2500 constraints. Restrictions you experience with the demo license, i.e. additional limits enforced by some solvers, time-limitation of 12 months, and a recent GAMS system also apply to the community license. The free demo and community license cannot be combined with a professional license. These free demo and community licenses are for demonstration and evaluation but not for commercial and production work.

4.10.3 GAMS Licenses

GAMS licenses beyond the free demo and community license require a license for the GAMS Base Module and for most of the GAMS/Solvers and the GAMS/Solver-Links:

- The GAMS Base Module includes the GAMS Language Compiler, GAMS-APIs, many utilities, some solvers without size restrictions, and most solvers in “demo” mode with a model size limitation. See details below.
- A GAMS/Solver connects the GAMS Base module to a particular solver and includes a license for this solver to be used through GAMS. It is not necessary to install additional software. A GAMS/Solver-Link connects the GAMS Base Module to a particular solver, but does not include a license for the solver. It may be necessary to install additional software before the solver can be used.

We distinguish between academic and commercial licenses. Academic licenses can only be used for teaching and research at degree granting institutions. Moreover, we distinguish between time-limited and perpetual licenses. For perpetual licenses the first year of maintenance, support and updates is included in the initial purchase of the software. Maintained licenses qualify for free updates, adding components, platform-switching without additional charge, and multi-copy discounts on the same platform. After the first year maintenance can be extended by paying a fee. Licenses are typically limited to a single computer platform type (e.g. Windows or Linux), but there are also licenses available, which will work on all major platform supported by GAMS. Free academic licenses are available for certain solvers. Please visit the GAMS website for further information about the pricing of an appropriate license.
4.10.4 Installing or updating a license file

A GAMS license file is an ASCII file of six lines, which was sent to you via e-mail. Please copy all six lines into a file gamslice.txt and place this into a location searched by GAMS. Details can be found in the installation notes of the specific operating system:

- GAMS Installation Notes for Windows
- GAMS Installation Notes for Mac OS X
- GAMS Installation Notes for Unix

Before installing a new GAMS version, please check whether it will work with your current GAMS license. The utility CHK4UPD can be used to see if a newer GAMS version is available and what the newest version is that can be used with a given license. This is done by running CHK4UPD without arguments or by giving the name of a local license file. CHK4UPD is also available through GAMS Studio and can be accessed through: Help -> Check for Update. Below is a sample output for a license, which is still under maintenance:

D:\>chk4upd
*** Processing GAMS system directory C:\gams\win64\24.8
*** Reading license file C:\ProgramData\GAMS\gamslice.txt
*** Your system is up to date
D:\>

If your license is no longer under maintenance, you will receive a message like the one below:

*** Processing GAMS system directory C:\GAMS\win64\23.9
*** Reading license file C:\ProgramData\GAMS\gamslice.txt
*** Your license is 1276 days too old to run with the most recent system
*** The version of your GAMS system is 23.9.5
*** The last GAMS version you can use is 24.1.3
*** Please request the download of an older version from sales@gams.com
*** For ordering an update to use the most recent version (24.8.5)
*** please contact your distributor

4.10.5 License Problems

Errors or warnings triggered by a problem with the GAMS license file (gamslice.txt) are reported both in the log file and in the lst file. Below are some typical error and warning messages and instructions how to overcome them.

4.10.5.1 No License File present

Without a license file GAMS will not work. If you have received a license file (gamslice.txt), follow the instruction from the previous section. Do not try to rename it or to modify the contents of the license file. Without a license GAMS will give you an error message like the one below:

*** No license specified and no gamslice.txt found in standard locations.
4.10.5.2 License File Invalid or Corrupted

Running GAMS with an invalid license will give you an error message like the one below:

*** License File validation failed

Do not try to modify your license file, any change will break it. If the original license file is no longer available, please contact sales@gams.com and ask for a copy of your license file. Please attach the license file you are currently using. This error also pops up, if you are using a license file, which was issued for a different platform.

4.10.5.3 License File expired

If are trying to run an old license file with a newer GAMS distribution, you may get an error message stating:

*** License file too old for this version of GAMS.
*** Maintenance expired 2840 days ago.
*** More than 60 days since expiration, sorry...
*** Run an older GAMS system or renew the license

Replace that GAMS system with the one you got together with your license file (use the CHK4UPD utility mentioned above to find out the latest version you can use with the current license; old systems are available on request from sales@gams.com) or update your license to the current version.

4.10.5.4 No License (for a particular Solver) found

If one of the solvers you are trying to use (or is selected as the default solver) is not included in your license file and the size of the model exceed the limits of the free demo or community version, you will get an error stating:

*** No license found
*** To update your license, please contact your distributor.

Check whether the default solvers for a particular problem class are included in the license. Please note that some of the solvers call other solvers as a sub-solver, e.g.:

- GAMS/DICOPT requires an NLP and a MIP solver
- GAMS/SBB requires an NLP solver
- MPSGE: If GAMS/PATH is not included in your license, please select GAMS/MILES, which is included in the base system as the default solver for MCP problems.

4.10.6 Warnings

4.10.6.1 License file too old for this version of GAMS

The complete message is:

*** License file too old for this version of GAMS.
*** Maintenance expired xx days ago.

If GAMS continues working, you can ignore this warning. However, any forthcoming GAMS distribution will most likely not work, but downgrade to the limits of the free demo system.
4.10.6.2 License File has expired xx days ago

This warning (without further error messages) indicates that the time limited license (e.g. an evaluation license) will stop working soon.

4.10.7 The GAMS/BASE Module

The GAMS/BASE module includes:

- **Language Compiler and Execution System**
- **GAMS IDE** *(Integrated Development Environment) (Windows only)*
  - Project Management
  - Editor, Syntax coloring, Spell checking
  - Launching and monitoring of (multiple) GAMS processes
  - Listing file / Tree view / Syntax-error navigation
  - Solver selection / option selection
  - GDX Viewer: Data cube and export (e.g. to MS Excel), charting facilities

- **GAMS Studio** *(platform independent, Beta version)*
  - Editor, Syntax coloring
  - Launching and monitoring of (multiple) GAMS processes
  - Listing file / Tree view / Syntax-error navigation
  - GDX Viewer

- **GAMS Free Solvers and Links**
  - Open Source (COIN-OR): CBC, Ipopt, SHOT
  - CONVERT, JAMS and LOGMIP, NLPEC
  - MILES
  - EXAMINER, GAMSCHK
  - Academic licenses only: ODHeuristic (requires a GAMS/CPLEX or GAMS/CPLEX-Link license), SCIP, Soplex
  - OSI Solver Links
  - GAMS/KESTREL for using the NEOS Server with a local GAMS system

- **Most other solvers in limited versions for demonstration and evaluation, but no commercial use:** 2000 constraints and variables for linear models, and 1000 constraints and variables for other model types. For licensed academic users the size limits are increased: 5000 constraints and variables for linear models, and 2500 constraints and variables for other model types. The solvers might enforce additional limits.

- **EMP** *(Extended Mathematical Programming Framework)*

- **Posix Utilities**
  - Academic licenses only: MIRO Connector for using external input and output symbols.

- **GDX** *(GAMSData eXchange) and related GDX Utilities*
  - Binary data exchange between application, GAMS, and the solver (fast, saves disk space, tailored for large sparse matrices, platform independent, direct GDX interfaces, API support for high-level programming languages)
4.10 Licensing

- **GDX Utilities:** GDX Viewer, GDXRANK, GDX2HAR, HAR2GDX, GDXMERGE, GDXDUMP, GDXCOPY, GDXDIFF, MDB2GMS, GDXMRW, GDXRW, GDXXRW, GDX2XLS, XLSDump, INVERT, CHOLESKY, EIGENVALUE, EIGENVECTOR

- **Various other tools**

- **GAMS APIs**
  - **Expert-Level GAMS APIs**
    - GAMS
    - GDX
    - Option
    - Supported Languages: C, C++, C#, Delphi, Fortran, Java, Python, VBA, VB.Net
  - **Object-Oriented GAMS APIs**
    - Supported Languages: C++, C#, Java, Python, VB.Net

- **Documentation**

- **Model Libraries**

### 4.10.8 Additional Limits for the Demo and Community License

The model size limits enforced by GAMS are:

- 2000 variables and 2000 constraints for linear (LP, RMIP, and MIP) models with a demo license
- 1000 variables and 1000 constraints for all other model types with a demo license
- 5000 variables and 5000 constraints for linear (LP, RMIP, and MIP) models with a community license
- 2500 variables and 2500 constraints for all other model types with a community license

In addition to the GAMS model size limits, the solvers might impose stricter limits when running with a demo or community license. We use $m$ for the number constraints, $n$ for the number variables, $nd$ for the number of discrete variables, $nz$ for the number nonzeros, and $nlnz$ for the number nonlinear nonzeros:

- **Antigone, Baron, LindoGlobal, and Octeract** require $m \leq 50$, $n \leq 50$, and $nlnz \leq 50$ with a demo license.
- **Antigone, Baron, LindoGlobal, and Octeract** require $m \leq 300$, $n \leq 300$, and $nlnz \leq 100$ with a community license.
- **Cplex and ODHCPLEX** require $m \leq 2000$ and $n \leq 2000$ with a *pro-bono* community license.
- **Decis and Knitro** require $m \leq 300$, $n \leq 300$, $nd \leq 50$, $nz \leq 2000$, and $nlnz \leq 1000$ with a demo and community license.
- **Xpress** requires $m + n \leq 5000$ with a community license.
- In addition **Gurobi** requires a license file from the vendor to operate.
4.11 A GAMS Tutorial by Richard E. Rosenthal

4.11.1 Introduction

Richard E. Rosenthal of the Naval Postgraduate School in Monterey, California wrote a detailed example of the use of GAMS for formulating, solving, and analyzing a small and simple optimization problem. The example is a quick overview of GAMS and its main features. Many references are made to other parts of the documentation, but they are only to tell you where to look for more details; the material here can be read profitably without reference to the rest of the documentation.

The example is an instance of the transportation problem of linear programming, which has historically served as a 'laboratory animal' in the development of optimization technology. [See, for example, Dantzig (1963)\(^1\).] It is a good choice for illustrating the power of algebraic modeling languages like GAMS because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices:

\[ i = \text{plants} \]
\[ j = \text{markets} \]

Given Data:

\[ a_i = \text{supply of commodity of plant } i \text{ (in cases)} \]
\[ b_j = \text{demand for commodity at market } j \]
\[ c_{ij} = \text{cost per unit shipment between plant } i \text{ and market } j \]

Decision Variables:

\[ x_{ij} = \text{amount of commodity to ship from plant } i \text{ to market } j \]

where \( x_{ij} \geq 0 \), for all \( i, j \)

Constraints:

- Observe supply limit at plant \( i \): \( \sum_j x_{ij} \leq a_i \) for all \( i \) (cases)
- Satisfy demand at market \( j \): \( \sum_i x_{ij} \geq b_j \) for all \( j \) (cases)

Objective Function: Minimize \( \sum_i \sum_j c_{ij} x_{ij} \) (\$K)

---

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol $\$K$ here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms *exogenous variable* and *endogenous variable* for *given data* and *decision variable*, respectively. In GAMS, the terminology adopted is as follows: indices are called *sets*, given data are called *parameters*, decision variables are called *variables*, and constraints and the objective function are called *equations*.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by a computer.

**Table 1**: Data for the transportation problem (adapted from Dantzig, 1963) illustrates Shipping Distances from Plants to Markets (1000 miles) as well as Market Demands and Plant Supplies.

<table>
<thead>
<tr>
<th>Plants ↓</th>
<th>New York</th>
<th>Chicago</th>
<th>Topeka</th>
<th>Markets ←</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
<td>350</td>
</tr>
<tr>
<td>San Diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
<td>600</td>
</tr>
</tbody>
</table>

Demands →
<table>
<thead>
<tr>
<th>Supplies ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>325</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>275</td>
</tr>
</tbody>
</table>

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the data given in table Table 1. Shipping distances are in thousands of miles, and shipping costs are assumed to be $90.00 per case per thousand miles. The GAMS representation of this problem is as follows:

```
$title a transportation model
Sets
   i     canning plants / seattle, san-diego /
   j     markets       / new-york, chicago, topeka / ;
Parameters
   a(i) capacity of plant i in cases
       / seattle  350
       san-diego 600 /
   b(j) demand at market j in cases
       / new-york 325
       chicago 300
       topeka 275 /
Table d(i,j) distance in thousands of miles
   seattle     new-york | chicago | topeka
   san-diego   2.5       | 1.7     | 1.8
   2.5         | 1.8     | 1.4 ;
Scalar f freight in dollars per case per thousand miles /90/ ;
Parameter c(i,j) transport cost in thousands of dollars per case ;
   c(i,j) = f * d(i,j) / 1000 ;
Variables
```
\( x(i,j) \) shipment quantities in cases
\( z \) total transportation costs in thousands of dollars;

Positive Variable \( x \);

Equations

\begin{align*}
\text{cost} & \quad \text{define objective function} \\
\text{supply}(i) & \quad \text{observe supply limit at plant } i \\
\text{demand}(j) & \quad \text{satisfy demand at market } j;
\end{align*}

\begin{align*}
\text{cost} & \quad z = \text{e}= \sum((i,j), c(i,j) \times x(i,j)) ; \\
\text{supply}(i) & \quad \sum(j, x(i,j)) = \text{l} = a(i) ; \\
\text{demand}(j) & \quad \sum(i, x(i,j)) = \text{g} = b(j) ;
\end{align*}

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

If you submit a file containing the statements above as input to GAMS, the transportation model will be formulated and solved. Details vary on how to invoke GAMS on different of computers, but the simplest ('no frills') way to call GAMS is to enter the word GAMS followed by the input file's name. You will see a number of terse lines describing the progress GAMS is making, including the name of the file onto which the output is being written. When GAMS has finished, examine this file, and if all has gone well the optimal shipments will be displayed at the bottom as follows.

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>50.000</td>
<td>300.000</td>
<td></td>
</tr>
<tr>
<td>san-diego</td>
<td>275.000</td>
<td>275.000</td>
<td></td>
</tr>
</tbody>
</table>

You will also receive the marginal costs (simplex multipliers) below.

<table>
<thead>
<tr>
<th></th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>0.036</td>
<td></td>
</tr>
<tr>
<td>san-diego</td>
<td>0.009</td>
<td></td>
</tr>
</tbody>
</table>

These results indicate, for example, that it is optimal to send nothing from Seattle to Topeka, but if you insist on sending one case it will add $0.036 \times 1000 = 36.00$ to the optimal cost.

4.11.2 Structure of a GAMS Model

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are listed in table Table 2.

\textbf{Table 2:} The basic components of a GAMS model
There are optional input components, such as edit checks for bad data and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

1. A GAMS model is a collection of statements in the GAMS Language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.

2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples in this tutorial, but precise rules of the road are given in the next Chapter.

3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The GAMS compiler does not distinguish between upper-and lowercase letters, so you are free to use either.

4. Documentation is crucial to the usefulness of mathematical models. It is more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more important, documentary text can be inserted within specific GAMS statements.

5. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration and an assignment or definition. *Declaration* means declaring the existence of something and giving it a name. *Assignment or definition* means giving something a specific value or form. In the case of equations, you must make the declaration and definition in separate GAMS statements. For all other GAMS entities, however, you have the option of making declarations and assignments in the same statement or separately.

6. The names given to the entities of the model must start with a letter and can be followed by up to 62 more letters or digits.
4.11.3 Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The Transportation example above contains just one Set statement:

```gams
Sets
   i canning plants / seattle, san-diego /;
   j markets / new-york, chicago, topeka /;
```

The effect of this statement is probably self-evident. We declared two sets and gave them the names `i` and `j`. We also assigned members to the sets as follows:

```gams
i = {Seattle, San Diego}
```

```gams
j = {New York, Chicago, Topeka}.
```

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes `/` rather than curly braces `{}' to delineate the set simply because not all computer keyboards have keys for curly braces. Note also that multiword names like 'New York' have to be quoted ((e.g. 'New York' or "New York") or you must use hyphens instead of blanks (e.g. New-York').

The words after the set names in the `sets` statement above are called text. Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and 'parrots' it back to you at various times for your convenience.

It was not necessary to combine the creation of sets `i` and `j` in one statement. We could have put them into separate statements as follows:

```gams
Set i canning plants / seattle, san-diego /;
Set j markets / new-york, chicago, topeka /;
```

The placement of blank spaces and lines (as well as the choice of upper- or lowercase) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular `set` is also up to you. Using `set` in a statement that makes a single declaration and `sets` in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid set statements in GAMS.

```gams
Set t time periods /1991*2000/;
Set m machines /mach1*mach24/;
```

Here the effect is to assign

```gams
m = {mach 1, mach 2,......, mach 24 }.
```

Note that set elements are stored as character strings, so the elements of `t` are not numbers.

Another convenient feature is the alias statement, which is used to give another name to a previously declared set. In the following example:

```gams
Alias (t,tp);
```

the name `tp` is like a `t' in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets `i`, `j`, `t`, and `m` in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in chapter `Dynamic Sets`. Another valuable advanced feature is multidimensional sets, which are discussed in section `Multi-Dimensional Sets`. 
4.11.4 Data

The GAMS model of the transportation problem demonstrates three of the different formats that are allowable for entering data. The three formats are:

- Lists
- Tables
- Direct assignments

The next three subsections will discuss each of these formats in turn.

4.11.4.1 Data Entry by Lists

The first format is illustrated by the first Parameters statement of the example, which is repeated below.

Parameters

a(i) capacity of plant i in cases
    / seattle 350
    san-diego 600 /

b(j) demand at market j in cases
    / new-york 325
    chicago 300
    topeka 275 / ;

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names $a$ and $b$, and declares their domains to be $i$ and $j$, respectively. (A domain is the set, or tuple of sets, over which a parameter, variable, or equation is defined.) The statement also gives documentary text for each parameter and assigns values of $a(i)$ and $b(j)$ for each element of $i$ and $j$. It is perfectly acceptable to break this one statement into two, if you prefer, as follows.

Parameters a(i) capacity of plant i in cases
    / seattle 350
    san-diego 600 / ;

Parameters b(j) demand at market j in cases
    / new-york 325
    chicago 300
    topeka 275 / ;

Here are some points to remember when using the list format.

1. The list of domain elements and their respective parameter values can be laid out in almost any way you like. The only rules are that the entire list must be enclosed in slashes and that the element-value pairs must be separated by commas or entered on separate lines.
2. There is no semicolon separating the element-value list from the name, domain, and text that precede it. This is because the same statement is being used for declaration and assignment when you use the list format. (An element-value list by itself is not interpretable by GAMS and will result in an error message.)

3. The GAMS compiler has an unusual feature called domain checking, which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell 'Seattle' correctly in the statement declaring Set i but misspell it as 'Seatle' in a subsequent element-value list, the GAMS compiler would give you an error message that the element 'Seatle' does not belong to the set i.

4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order.

5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a Scalar statement containing a degenerate list of only one value, as in the following statement from the transportation model:

Scalar f freight in dollars per case per thousand miles /90/;

If the domain of a parameter has two or more dimensions, it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few non-zeros) and super-sparse (having few distinct non-zeros).

4.11.4.2 Data Entry by Tables

Optimization practitioners have noticed for some time that many of the input data for a large model are derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided in the transportation model:

Table d(i,j) distance in thousands of miles

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

The effect of this statement is to declare the parameter d and to specify its domain as the set of ordered pairs in the Cartesian product of i and j. The values of d are also given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in Chapter Data Entry: Parameters, Scalars and Tables.

4.11.4.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method:

Parameter c(i,j) transport cost in thousands of dollars per case ;

\[ c(i,j) = f \times d(i,j) / 1000 ; \]
It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you an error message.)

The effects of the first statement above are to declare the parameter \( c \), to specify the domain \((i,j)\), and to provide some documentary text. The second statement assigns to \( c(i,j) \) the product of the values of the parameters \( f \) and \( d(i,j) \). Naturally, this is legal in GAMS only if you have already assigned values to \( f \) and \( d(i,j) \) in previous statements.

The direct assignment above applies to all \((i,j)\) pairs in the domain of \( c \). If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

\[
c('Seattle','New-York') = 0.40;
\]

is a valid GAMS assignment statement.

The same parameter can be assigned a value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as C, you will have no trouble in becoming comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies not shared by C. For example, we were able to assign \( c(i,j) \) values for all \((i,j)\) pairs without constructing 'loops'.)

The GAMS standard operations and supplied functions are given later. Here are some examples of valid assignments. In all cases, assume the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values in previous statements.

\[
\begin{align*}
\text{csquared} &= \text{sqr}(c); \\
\text{e} &= m*\text{csquared}; \\
\text{w} &= 1/\text{lamda}; \\
\text{eqo}(i) &= \text{sqrt}(2*\text{demand}(i)\*\text{ordcost}(i)/\text{holdcost}(i)); \\
\text{t}(i) &= \min(p(i), q(i)/r(i), \log(s(i))); \\
\text{euclidean}(i,j) &= \text{qrt}(\text{sqr}(\text{xi}(i) - \text{xi}(j) + \text{sqr}(\text{x2}(i) - \text{x2}(j)))); \\
\text{present}(j) &= \text{future}(j)*\exp(-\text{interest}\*\text{time}(j));
\end{align*}
\]

The summation and product operators to be introduced later can also be used in direct assignments.

### 4.11.5 Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a `Variables` statement. Each variable is given a name, a domain if appropriate, and (optionally) text. The transportation model contains the following example of a `Variables` statement.

Variables

\[
\begin{align*}
\text{x}(i,j) & \quad \text{shipment quantities in cases} \\
\text{z} & \quad \text{total transportation costs in thousands of dollars}
\end{align*}
\]

This statement results in the declaration of a shipment variable for each \((i,j)\) pair. (You will see in chapter Equations, how GAMS can handle the typical real-world situation in which only a subset of the \((i,j)\) pairs is allowable for shipment.)

The \( z \) variable is declared without a domain because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. A selection of permissible types are given in table Table 3. For a full list, see section Variable Types.

**Table 3:** Permissible variable types
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Allowed Range of Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>free (default)</td>
<td>$-\infty$ to $+\infty$</td>
</tr>
<tr>
<td>positive</td>
<td>0 to $+\infty$</td>
</tr>
<tr>
<td>negative</td>
<td>$-\infty$ to 0</td>
</tr>
<tr>
<td>binary</td>
<td>0 or 1</td>
</tr>
<tr>
<td>integer</td>
<td>0, 1, ..., 100 (default)</td>
</tr>
</tbody>
</table>

The variable that serves as the quantity to be optimized must be a scalar and must be of the free type. In our transportation example, $z$ is kept free by default, but $x(i,j)$ is constrained to non-negativity by the following statement.

Positive variable $x$ ;

Note that the domain of $x$ should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

Section The .lo, .l, .up, .m Database describes how to assign lower bounds, upper bounds, and initial values to variables.

4.11.6 Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities that comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

4.11.6.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, Equations in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation model contains the following equation declaration:

Equations
cost define objective function
supply(i) observe supply limit at plant i
demand(j) satisfy demand at market j ;

Keep in mind that the word Equation has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and a GAMS equation with a single name can refer to one or several of these relationships. For example, cost has no domain so it is a single equation, but supply refers to a set of inequalities defined over the domain $i$. 

4.11.6.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe the summation notation in GAMS. Remember that
GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would
it be convenient for the user) to employ the standard mathematical notation for summations.

The summation notation in GAMS can be used for simple and complex expressions. The format is
based on the idea of always thinking of a summation as an operator with two arguments: \( \text{Sum(}
\text{index of summation, summand}) \) A comma separates the two arguments, and if the first argument
requires a comma then it should be in parentheses. The second argument can be any mathematical expression
including another summation.

As a simple example, the transportation problem contains the expression

\[ \text{Sum}(j, x(i,j)) \]

that is equivalent to \( \sum_j x_{ij} \).

A slightly more complex summation is used in the following example:

\[ \text{Sum}((i,j), c(i,j)*x(i,j)) \]

that is equivalent to \( \sum_i \sum_j c_{ij}x_{ij} \).

The last expression could also have been written as a nested summation as follows:

\[ \text{Sum}(i, \text{Sum}(j, c(i,j)*x(i,j))) \]

In section Conditional Indexed Operations, we describe how to use the dollar operator to impose restric-
tions on the summation operator so that only the elements of \( i \) and \( j \) that satisfy specified conditions are
included in the summation.

Products are defined in GAMS using exactly the same format as summations, replacing \( \text{Sum} \) by \( \text{Prod} \). For
example,

\[ \text{prod}(j, x(i, j)) \]

is equivalent to: \( \Pi_j x_{ij} \).

Summation and product operators may be used in direct assignment statements for parameters. For
example,

\[ \text{scalar totsupply total supply over all plants;} \]
\[ \text{totsupply} = \text{sum}(i, a(i)); \]
4.11.6.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of their variety. The components of an equation definition are, in order:

1. The name of the equation being defined
2. The domain
3. Domain restriction condition (optional)
4. The symbol ' .. '
5. Left-hand side expression
6. Relational operator: =l=, =e=, =g= or others. For a complete list, see Table Equation Types.
7. Right-hand side expression

The transportation example contains three of these statements.

```
cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;
```

Here are some points to remember.

- The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the definition for the demand constraint will result in the creation of one constraint for each element of the domain j, as shown in the following excerpt from the GAMS output.

```
demand(new-york)..x(seattle,new-york) + x(san-diego,new-york)=g=325 ;
```  
```
demand(chicago).. x(seattle,chicago) + x(san-diego,chicago) =g=300 ;
```  
```
demand(topeka).x(seattle,topeka) + x(san-diego,topeka) =g=275 ;
```

- The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)

- In many real-world problems, some of the members of an equation domain need to be omitted or differentiated from the pattern of the others because of an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as the dollar or 'such-that' operator, which is not illustrated here. The domain restriction feature can be absolutely essential for keeping the size of a real-world model within the range of solvability.

- The relational operators have the following meanings:
  
  - =l=  less than or equal to
  - =g=  greater than or equal to
  - =e=  equal to

- It is important to understand the difference between the symbols '=' and '=e='. The '=' symbol is used only in direct assignments, and the '=e=' symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definition also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.
4.11 A GAMS Tutorial by Richard E. Rosenthal

- Variables can appear on the left or right-hand side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.

- An equation definition may appear anywhere in the GAMS input, provided the equation and all variables and parameters to which it refers were previously declared. (Note that it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they were declared.

4.11.7 Objective Function

This is just a reminder that GAMS has no explicit entity called the objective function. To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain) and which appears in an equation definition that equates it to the objective function.

4.11.8 Model and Solve Statements

The word model has a very precise meaning in GAMS. It is simply a collection of equations. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the keyword model followed by the name of the model, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter \texttt{/all/} in place of the explicit list. In our example, there is one model statement:

\begin{verbatim}
model transport /all/ ;
\end{verbatim}

This statement may seem superfluous, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut \texttt{/all/}, the statement would be written as

\begin{verbatim}
model transport / cost, supply, demand / ;
\end{verbatim}

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprises a specific model (or sub-model) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a \texttt{solve} statement, which in our example is written as

\begin{verbatim}
solve transport using lp minimizing z ;
\end{verbatim}

The format of the solve statement is as follows:

1. The key word solve
2. The name of the model to be solved
3. The key word using
4. An available solution procedure. A complete list is given below. For further details, see section Classification of Models.
<table>
<thead>
<tr>
<th>Solution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp</td>
<td>for linear programming</td>
</tr>
<tr>
<td>qcp</td>
<td>for quadratic constraint programming</td>
</tr>
<tr>
<td>nlp</td>
<td>for nonlinear programming</td>
</tr>
<tr>
<td>dnlp</td>
<td>for nonlinear programming with discontinuous derivatives</td>
</tr>
<tr>
<td>mip</td>
<td>for mixed integer programming</td>
</tr>
<tr>
<td>rmip</td>
<td>for relaxed mixed integer programming</td>
</tr>
<tr>
<td>miqcp</td>
<td>for mixed integer quadratic constraint programming</td>
</tr>
<tr>
<td>rmiqcp</td>
<td>for relaxed mixed integer quadratic constraint programming</td>
</tr>
<tr>
<td>minlp</td>
<td>for mixed integer nonlinear programming</td>
</tr>
<tr>
<td>rminlp</td>
<td>for relaxed mixed integer nonlinear programming</td>
</tr>
<tr>
<td>mcp</td>
<td>for mixed complementarity problems</td>
</tr>
<tr>
<td>mpec</td>
<td>for mathematical programs with equilibrium constraints</td>
</tr>
<tr>
<td>rmpec</td>
<td>for relaxed mathematical program with equilibrium constraints</td>
</tr>
<tr>
<td>cns</td>
<td>for constrained nonlinear systems</td>
</tr>
<tr>
<td>emp</td>
<td>for extended mathematical programming</td>
</tr>
</tbody>
</table>

5. The keyword 'minimizing' or 'maximizing'

6. The name of the variable to be optimized

### 4.11.9 Display Statements

The `solve` statement will cause several things to happen when executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the output from the solver will be printed to a file. To get the optimal values of the primal and/or dual variables, we can look at the solver output, or, if we wish, we can request a display of these results from GAMS. Our example contains the following statement:

```gams
display x.l, x.m ;
```

that calls for a printout of the final levels, `x.l`, and marginal (or reduced costs), `x.m`, of the shipment variables, `x(i,j)`. GAMS will automatically format this printout in to dimensional tables with appropriate headings.

### 4.11.10 The .lo, .l, .up, .m Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. The most important fields in each record are:

- `.lo` lower bound
- `.l` level or primal value
- `.up` upper bound
- `.m` marginal or dual value

The format for referencing these quantities is the variable or equation's name followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read-and write-access to the database. This may not seem remarkable to you now, but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.
4.11.10.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (free, positive, negative, binary, or integer), but these bounds can be overwritten by the GAMS user. Some examples follow.

\[
\begin{align*}
x.\text{up}(i,j) &= \text{capacity}(i,j) ; \\
x.\text{lo}(i,j) &= 10.0 ; \\
x.\text{up}(\text{seattle},\text{new-york}) &= 1.2*\text{capacity}(\text{seattle},\text{new-york}) ;
\end{align*}
\]

It is assumed in the first and third examples that \( \text{capacity}(i,j) \) is a parameter that was previously declared and assigned values. These statements must appear after the variable declaration and before the \texttt{Solve} statement. All the mathematical expressions available for direct assignments are usable on the right-hand side.

In nonlinear programming it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bound. It is also very helpful to specify an initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are \( \text{quantity}(i) \), and it is known that the optimal solution to the unconstrained version of the problem is a parameter called \( \text{eoq}(i) \). As a guess for the optimum of the constrained problem we enter

\[
\text{quantity.l}(i) = 0.5*\text{eoq}(i) ;
\]

(The default initial level is zero unless zero is not within the bounded range, in which case it is the bound closest to zero.)

It is important to understand that the \texttt{.lo} and \texttt{.up} fields are entirely under the control of the GAMS user. The \texttt{.l} and \texttt{.m} fields, in contrast, can be initialized by the user but are then controlled by the solver.

4.11.10.2 Transformation and Display of Optimal Values

(This section may be skipped on first reading if desired.)

After the optimizer is called via the \texttt{solve} statement, the values it computes for the primal and dual variables are placed in the database in the \texttt{.l} and \texttt{.m} fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the solve statement, we would enter

\[
\begin{align*}
\text{parameter pctx(i,j) perc of market j's demand filled by plant i;} \\
pctx(i,j) &= 100.0*x.l(i,j)/b(j) ; \\
\text{display pctx ;}
\end{align*}
\]

Appending these commands to the original transportation problem input results in the following output:

\[
\begin{array}{llll}
pctx & \text{percent of market j's demand filled by plant I} \\
\text{new-york} & \text{chicago} & \text{topeka} \\
\text{seattle} & 15.385 & 100.000 \\
\text{san-diego} & 84.615 & 100.000
\end{array}
\]
For an example involving marginal, we briefly consider the *ratio constraints* that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let \( y(i,j) \) be the variable for the number of tons of raw material \( i \) put into finished product \( j \). Suppose the *ratio constraint* is that no product can consist of more than 25 percent of one ingredient, that is,

\[
y(i,j)/q(j) = l= 0.25 ;
\]

for all \( i, j \). To keep the model linear, the constraint is written as

\[
\text{ratio}(i,j) .. y(i,j) - 0.25*q(j) = l= 0.0 ;
\]

rather than explicitly as a ratio.

The problem here is that \( \text{ratio}.m(i,j) \), the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

\[
y(i,j) - 0.25*q(j) = l= 1.0 ;
\]

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ratio constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

\[
y(i,j)/q(j) = l= 0.26 ;
\]

We can in fact obtain the desired marginals by entering the following transformation on the undesired marginals:

```plaintext
parameter amr(i,j) appropriate marginal for ratio constraint ;
amr(i,j) = ratio.m(i,j)*0.01*q.l(j) ;
display amr ;
```

Notice that the assignment statement for \( \text{amr} \) accesses both \( .m \) and \( .l \) records from the database. The idea behind the transformation is to notice that

\[
y(i,j)/q(j) = l= 0.26 ;
\]

is equivalent to

\[
y(i,j) - 0.25*q(j) = l= 0.01*q(j) ;
\]
4.11.11 GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion, see chapter GAMS Output. This tutorial discusses output partially as follows:

- Echo Print
- Reference Maps
- Status Reports
- Error Messages
- Model Statistics
- Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the false impression that flawless use of advanced software should be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users will make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

4.11.11.1 Echo Prints

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

```plaintext
2 Sets
3    i   canning plants       / seattle, san-diego /
4    j   markets             / new-york, chicago, topeka /
5
6 Parameters
7
8    a(i)  capacity of plant i in cases
9        /  seattle  350
10       san-diego  600 /
11
12    b(j)  demand at market j in cases
13        /  new-york  325
14       chicago  300
15       topeka   275 /
16
17 Table d(i,j)  distance in thousands of miles
18    new-york  chicago  topeka
19    seattle  2.5    1.7    1.8
20    san-diego  2.5    1.8    1.4 ;
21
22 Scalar f  freight in dollars per case per thousand miles /90/ ;
23
24 Parameter c(i,j)  transport cost in thousands of dollars per case ;
25
26    c(i,j) = f * d(i,j) / 1000 ;
27
28 Variables
```
29 \( x(i,j) \) shipment quantities in cases
30 \( z \) total transportation costs in thousands of dollars ;
31
32 Positive Variable \( x \) ;
33
34 Equations
35 \( \text{cost} \) define objective function
36 \( \text{supply}(i) \) observe supply limit at plant \( i \)
37 \( \text{demand}(j) \) satisfy demand at market \( j \) ;
38
39 \( \text{cost} .. \quad z =e= \sum(i,j), c(i,j)\times x(i,j) ; \)
40
41 \( \text{supply}(i) .. \quad \sum(j, x(i,j)) =l= a(i) ; \)
42
43 \( \text{demand}(j) .. \quad \sum(i, x(i,j)) =g= b(j) ; \)
44
45 Model transport /all/ ;
46
47 Solve transport using lp minimizing \( z \) ;
48
49 Display \( x.l, \) \( x.m \) ;
50

The reason this echo print starts with line number 2 rather than line number 1 is because the input file contains a dollar-print-control statement. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1.

\$title a transportation model

The \$title statement causes the subsequent text to be printed at the top of each page of output. Other available instructions are given in chapter Dollar Control Options.

4.11.11.2 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with **** and contain a '$' directly below the point at which the compiler thinks the error occurred. The $ is followed by a numerical error code, which is explained after the echo print. Several examples follow.

Example 1: Entering the statement

\[ \text{set \ q quarterly time periods / spring, sos1, fall, wtr / ;} \]

results in the echo

\[ \text{1 \ set \ q quarterly time periods / spring, sos1, fall, wtr / ; } \quad **** \]

In this case, the GAMS compiler indicates that something is wrong with the set element sum. At the bottom of the echo print, we see the interpretation of error code 160:

Error Message
160 Unique element expected....
The problem is that sos1 is a reserved word, which cannot be used as identifiers in general. The complete list of reserved words is shown in this chapter. Thus our set element must have a unique name like 'summer'. This is a common beginner's error.

**Example 2:** Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of \( c(i,j) \), as follows.

```plaintext
Parameter c(i,j) transport cost in 1000s of dollars per case
c(i,j) = f * d(i,j) / 1000 ;
```

Here is the resulting output.

```
16 Parameter c(i,j) transport cost in 1000s of dollars per case
17 c(i,j) = f * d(i,j)/1000
**** $97 $195,96,409
Error Message
96 Blank needed between identifier and text
(-or- illegal character in identifier)
(-or- check for missing ';' on previous line)
97 Explanatory text can not start with '$', '=', or '..'
(-or- check for missing ';' on previous line)
195 Symbol redefined with a different type
409 Unrecognizable item - skip to find a new statement
looking for a ';' or a key word to get started again
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the other! The first error detected (in line 17), code 97, indicate that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16 rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.

Unfortunately, you cannot always expect error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the c entry in the cross-reference list (to be explained in the next section) and noticing that it was never assigned.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TYPE</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>PARAM</td>
<td>declared</td>
</tr>
</tbody>
</table>

**Example 3:** Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with 'Seattle' spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```
4 sets
5   i canning plants /seattle, san-diego /
6   j markets /new-york, chicago, topeka /
7
8 table d(i,j) distance in thousand of miles
9   new-york  chicago  topeka
10   seattle  2.5    1.7    1.8
11   san-diego 2.5    1.8    1.4
**** $170
Error Message
170 Domain violation for element
```
Example 4: Similarly, if we mistakenly enter dem(j) instead of b(j) as the right-hand side of the demand constraint, the result is

45  demand(j) ..  sum(i, x(i,j) ) =g=  dem(j) ;

****
   Error Message
  140  Unknown symbol

Example 5: The next example is a mathematical error, which is sometimes committed by novice modelers and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

For all $i$, $\sum_{j} x_{ij} = 100$

There are two errors in this equation, both having to do with the control of indices. Index $i$ is over-controlled and index $j$ is under-controlled.

You should see that index $i$ is getting conflicting orders. By appearing in the quantifier 'for all $i$', it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index $j$ is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

   meaningless(i) ..  sum(i, x(i,j)) =e=  100 ;

****
   Error Messages
  125  Set is under control already [This refers to set $i$.]
  149  Uncontrolled set entered as constant [This refers to set $j$.]

More information about error reporting is given in section Error Reporting. Comprehensive error detection and well-designed error messages are a big help in getting models implemented quickly and correctly.

4.11.11.3 Reference Maps

The next section of output, which is the last if errors have been detected, is a pair of reference maps that contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a cross-reference map such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables, and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows (we do not display all tables). To turn the cross-reference map on, please add the line

$onSymXRef

at the beginning of your model right after the $title statement.
For example, the cross-reference list tells us that the symbol \( a \) is a parameter that was declared in line 9, defined (assigned value) in line 10, and referenced in line 42. The symbol \( i \) has a more complicated entry in the cross-reference list. It is shown to be a set that was declared and defined in line 4. It is referenced once in lines 9, 18, 25, 27, 30, 37, 44 and referenced twice in lines 40 and 44. Set \( i \) is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 27, 40, 42 and 44.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model owing to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For example, this list is as follows.

**SETS**

\( i \)   canning plants  
\( j \)   markets

**PARAMETERS**

\( a \)   capacity of plant \( i \) in cases  
\( b \)   demand at market \( j \) in cases  
\( c \)   transport cost in thousands of dollars per case  
\( d \)   distance in thousands of miles  
\( f \)   freight in dollars per case per thousand miles

**VARIABLES**

\( x \)   shipment quantities in cases  
\( z \)   total transportation costs in thousands of dollars

**EQUATIONS**

\( \text{cost} \)   define objective function  
\( \text{demand} \)   satisfy demand at market \( j \)  
\( \text{supply} \)   observe supply limit at plant \( i \)

**MODELS**

\( \text{transport} \)
4.11.11.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer it, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question. A product of the solve command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

\[
\text{demand}(j) \ldots \sum(i, x(i,j)) \geq b(j) ;
\]

while the equation listing of specific constraints is

\[
\begin{align*}
\text{--------demand} & \quad =g= \quad \text{satisfy demand at market j} \\
\text{demand(new-york)} & \quad x(\text{seattle, new-york}) + x(\text{san-diego, new-york}) \geq 325 ; \\
\text{demand(chicago)} & \quad x(\text{seattle, chicago}) + x(\text{san-diego, chicago}) \geq 300 ; \\
\text{demand(topeka)} & \quad x(\text{seattle, topeka}) + x(\text{san-diego, topeka}) \geq 275 ;
\end{align*}
\]

The default output is a maximum of three specific equations for each generic equation. To change the default, insert an input statement prior to the solve statement:

\[\text{option limrow = r ;}\]

where \(r\) is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the above command can be extended:

\[\text{option limrow = r, limcol = c ;}\]

where \(c\) is the desired number of columns. (Setting \text{limrow} and \text{limcol} to 0 is a good way to reduce the size of your lst file after your model has been debugged.) In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

4.11.11.5 Model Statistics

The last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size, as shown below for the transportation example.

\begin{center}
\text{MODEL STATISTICS}
\end{center}

\begin{tabular}{|l|c|c|}
\hline
\text{BLOCKS OF EQUATIONS} & 3 & \text{SINGLE EQUATIONS} & 6 \\
\text{BLOCKS OF VARIABLES} & 2 & \text{SINGLE VARIABLES} & 7 \\
\text{NON ZERO ELEMENTS} & 19 & & \\
\hline
\end{tabular}

The \text{BLOCK} counts refer to the number of generic equations and variables. The \text{SINGLE} counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to describe the degree of non-linearity in the problem.
### 4.11.11.6 Status Reports

After the solver executes, GAMS prints out a brief *Solve Summary* whose two most important entries are SOLVER STATUS and the MODEL STATUS. For our transportation problem the solve summary is as follows:

```
SOLVE SUMMARY
MODEL TRANSPORT OBJECTIVE z
TYPE LP DIRECTION MINIMIZE
SOLVER CPLEX FROM LINE 49

**** SOLVER STATUS 1 Normal Completion
**** MODEL STATUS 1 Optimal
**** OBJECTIVE VALUE 153.6750
```

The status reports are preceded by the same `****` string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you look at an output file for the first time. The desired solver status is 1 NORMAL COMPLETION, but there are other possibilities, documented in section The Solve Summary, which relate to various types of errors and mishaps.

There are eleven possible model statuses, including the usual linear programming termination states (1 Optimal, 3 Unbounded, 4 Infeasible), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is 2 Locally Optimal. The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality. In integer programming, the status to look for is 8 Integer Solution. This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

### 4.11.11.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in as standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit, and marginal. Generic equation block and the column output group the row output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for `supply(i)`, `demand(j)`, and `x(i,j)` are as follows:

```
---- EQU supply observe supply limit at plant i
    LOWER  LEVEL  UPPER  MARGINAL
seattle  -INF  350.000  350.000  EPS
san-diego -INF  550.000  600.000  .

---- EQU demand satisfy demand at market j
    LOWER  LEVEL  UPPER  MARGINAL
```
The single dots '.' in the output represent zeroes. The entry EPS, which stands for epsilon, mean very small but nonzero. In this case, EPS indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with EPS rather than zero to facilitate restarting the optimizer from the old basis.)

If the solvers results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with INFES or NOPT, respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked UNBND.

At the end of the solvers solution report is a very important report summary, which gives a tally of the total number of non-optimal, infeasible, and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

**** REPORT SUMMARY : 0 NONOPT 0 INFEASIBLE 0 UNBOUNDED

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the .l and .m fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed, and GAMS automatically formats and labels an appropriate array. For example, the input statement.

display x.l, x.m ;

results in the following output.

---- 50 VARIABLE x.L shipment quantities in cases

new-york  325.000  325.000  +INF  0.225
chicago   300.000  300.000  +INF  0.153
topeka    275.000  275.000  +INF  0.126

---- VAR x shipment quantities in cases

 LOWER   LEVEL   UPPER   MARGINAL

seattle .new-york  .  50.000  +INF  .
seattle .chicago   .  300.000 +INF  .
seattle .topeka    .   .  +INF  0.036
san-diego .new-york .  275.000 +INF  .
san-diego .chicago  .   .  +INF  0.009
san-diego .topeka  .  275.000  +INF  .

As seen in reference maps, equation listings, solution reports, and optional displays, GAMS saves the documentary text and 'parrots' it back throughout the output to help keep the model well documented.
4.11.12 Summary

This tutorial has demonstrated several of the design features of GAMS that enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS versus a matrix generator or conversational solver.

- By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.

- Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.

- You save time and reduce generation errors by creating whole sets of closely related constraints in one statement.

- You can save time and reduce input errors by providing formulae for calculating the data rather than entering them explicitly.

- The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.

- The output of GAMS is easy to read and use. The solution report from the solver is automatically reformatted so that related equations and variables are grouped together and appropriately labeled. Also, the display command allows you to modify and tabulate results very easily.

- If you are teaching or learning modeling, you can benefit from the insistence of the GAMS compiler that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which errors are detected should greatly reduce development time.

- By using the dollar operator and other advanced features not covered in this tutorial, one can efficiently implement large-scale models. Specific applications of the dollar operator include:

  1. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem within the range of solvability.

  2. It can be used to build complex summations and products, which can then be used in equations or customized reports.

  3. It can be used for issuing warning messages or for terminating prematurely conditioned upon context-specific data edits.

4.12 Quick Start Tutorial

GAMS (the General Algebraic Modeling System) is a high-level modeling system for mathematical programming problems. This tutorial is aimed at modelers who are new to GAMS and are looking for a quick introduction to the core features of GAMS. Note that the text contains many links to the GAMS User's Guide for further exploration. Observe that this tutorial is adapted from the Quick Start Tutorial from Bruce McCarl. Another introductory text is the tutorial A GAMS Tutorial by Richard E. Rosenthal.
4.12.1 Three Basic Models

Most modelers who are interested in GAMS have one (or more) of the following backgrounds:

- They wish to solve constrained optimization problems with objective functions.
- They are looking for a system to solve general equilibrium problems that arise in various areas of economics.
- They wish to solve nonlinear systems of equations that arise in engineering.

GAMS is well equipped to model and solve all three types of problems. We will start by introducing examples for all three areas and the discussions that will follow will refer to these three basic examples.

4.12.1.1 First Example: Solving a Linear Programming Problem (LP)

The simplest constrained optimization problem is an LP. Assume we wish to solve the following LP that represents a simple farm model where profit is maximized:

\[
\begin{align*}
\text{max} & \quad 109X_{\text{corn}} + 90X_{\text{wheat}} + 115X_{\text{cotton}} \\
\text{s.t.} & \quad X_{\text{corn}} + X_{\text{wheat}} + X_{\text{cotton}} \leq 100 \quad \text{(land)} \\
& \quad 6X_{\text{corn}} + 4X_{\text{wheat}} + 8X_{\text{cotton}} \leq 500 \quad \text{(labor)} \\
& \quad X_{\text{corn}} \geq 0 \quad \text{(nonnegativity)} \\
& \quad X_{\text{wheat}} \geq 0 \quad \text{(nonnegativity)} \\
& \quad X_{\text{cotton}} \geq 0 \quad \text{(nonnegativity)}
\end{align*}
\]

Note that there are three decision variables: \(X_{\text{corn}}\) denotes the land assigned to growing corn, \(X_{\text{wheat}}\) is the land assigned to growing wheat and \(X_{\text{cotton}}\) represents the land where cotton is grown. In the first line the total profit is expressed as a function of the land allotted to the three crops, the multipliers represent the expected profit per acre depending of the type of crop. The first inequality imposes a limit on the available land, the second inequality imposes a limit on the available labor and the final three lines restrict the decision variables to nonnegative values.

This problem may be expressed in GAMS as follows:

```gams
Positive Variables Xcorn, Xwheat, Xcotton;
Variables Z;
Equations obj, land, labor;
obj.. Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land.. Xcorn + Xwheat + Xcotton =l= 100;
labor.. 6 * Xcorn + 4 * Xwheat + 8 * Xcotton =l= 500;
Model farmproblem / obj, land, labor /;
solve farmproblem using LP maximizing Z;
```

We will analyze this formulation and discuss every part of it after introducing the other two basic examples.
4.12 Quick Start Tutorial

4.12.1.2 Second Example: Solving for an Economic Equilibrium

The simplest general equilibrium model is the single good, single market problem. Assume we wish to solve the following equilibrium problem:

Demand Price: \[ P \geq P_d = 6 - 0.3Q_d \]

Supply Price: \[ P \leq P_s = 1 + 0.2Q_s \]

Quantity Equilibrium: \[ Q_s \geq Q_d \]

Nonnegativity: \[ P \geq 0 \]
\[ Q_s \geq 0 \]
\[ Q_d \geq 0 \]

Here \( P \) is the market clearing price, \( P_d \) is the demand curve, \( Q_d \) denotes the quantity demanded, \( P_s \) the supply curve and \( Q_s \) the quantity supplied. Note that this is a problem in 3 equations and 3 variables. The variables are \( P, Q_d, \) and \( Q_s \). Note that \( P_d \) and \( P_s \) are not variables, since they can be computed from the equality relations.

Usually all equality constraints are used for such a set up. However, we will use a more general setup, because it relaxes some assumptions and more accurately depicts a model ready to be implemented in GAMS. In particular, we permit the case where the supply curve price intercept may be above the demand curve price intercept and thus the market may clear with a nonzero price but a zero quantity. In addition, we allow the market price to be above the demand curve price and below the supply curve price. We also impose some additional conditions based on Walras’ Law to ensure a proper solution in such cases.

\[ Q_d(P - P_d) = 0 \text{ or } Q_d(P_d - (6 - 0.3Q_d)) = 0 \]
\[ Q_s(P - P_s) = 0 \text{ or } Q_s(P_s - (1 + 0.2Q_s)) = 0 \]
\[ P(Q_s - Q_d) = 0 \]

Here the quantity demanded is nonzero only if the market clearing price equals the demand curve price, the quantity supplied is nonzero only if the market clearing price equals the supply curve price and the market clearing price is only nonzero if \( Q_s = Q_d \).

The simplest GAMS formulation is given below. Note that we needed to rearrange the equation \( \text{Psupply} \) in order to achieve a greater than inequality, thus we accomodated the requirements of the solver PATH.

Positive Variables \( P, Q_d, Q_s \);

Equations \( \text{Pdemand, Psupply, Equilibrium;} \)
\( \text{Pdemand.} \quad P \implies g = 6 - 0.3\text{Qd;} \)
\( \text{Psupply.} \quad (1 + 0.2\text{Qs}) \implies g = P; \)
\( \text{Equilibrium..} \quad Q_s \implies g = Q_d; \)

Model problem / \text{Pdemand.Qd, Psupply.Qs, Equilibrium.P} /;

solve problem using MCP;

We will analyze this formulation and discuss every component of it after introducing the third example.
4.12.1.3 Third Example: Solving a Nonlinear Equation System

Engineers often wish to solve a nonlinear system of equations. Examples include chemical equilibria or problems in the context of oil refining. Many other such problems of this type exist. The problem that follows is adapted from the paper Wall, T W, Greening, D, and Woolsey , R E D, “Solving Complex Chemical Equilibria Using a Geometric-Programming Based Technique” Operations Research 34, 3 (1987).

\[ ba \ast so4 = 1 \]
\[ baoh / ba / oh = 4.8 \]
\[ hso4 / so4 / h = 0.98 \]
\[ h \ast oh = 1 \]
\[ ba + 1e-7*baoh = so4 + 1e-5*hso4 \]
\[ 2 \ast ba + 1e-7*baoh + 1e-2*h = 2 \ast so4 + 1e-5*hso4 + 1e-2*oh \]

Note that this is a nonlinear system of equations with the variables \( ba, so4, baoh, oh, hso4 \) and \( h \). The following formulation in GAMS is from the model [WALL] in the GAMS model library.

Variables \( ba, so4, baoh, oh, hso4, h \);

Equations \( r1, r2, r3, r4, b1, b2 \);

\[ r1.. \quad ba \ast so4 =e= 1 \]
\[ r2.. \quad baoh / ba / oh =e= 4.8 \]
\[ r3.. \quad hso4 / so4 / h =e= 0.98 \]
\[ r4.. \quad h \ast oh =e= 1 \]
\[ b1.. \quad ba + 1e-7*baoh =e= so4 + 1e-5*hso4 \]
\[ b2.. \quad 2 \ast ba + 1e-7*baoh + 1e-2*h =e= 2 \ast so4 + 1e-5*hso4 + 1e-2*oh \]

Model wall / all / ;

ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;
solve wall using nlp minimizing ba;

4.12.2 Components of the Example Models

Now that we have our three basic models in place, we will analyze them and identify and discuss their components, including variables, equations, model definitions, solve statements and starting points.

4.12.2.1 Variables in the Example Models

Variables in GAMS have to be declared as variables with a variable statement. The variable statements in the examples above are repeated below:

- First example:
4.12 Quick Start Tutorial

Positive Variables  Xcorn, Xwheat, Xcotton;
Variables  Z;

• Second example:
Positive Variables  P, Qd , Qs;

• Third example:
Variables  ba, so4, baoh, oh, hso4, h ;

Note that variable(s) is a keyword in GAMS. Positive is another keyword, it serves as a modifier to the keyword variable and has the effect that the variables that follow may take only nonnegative values. Variables that are declared with the keyword variable without a modifier are unrestricted in sign. For more details on variable declarations and variable types, see sections Variable Declarations and Variable Types respectively.

Observe that GAMS is not case sensitive and allows a line feed to be used instead of a comma. Thus, the following three variable declarations are all valid and have the same effect:

POSITIVE VARIABLES  Xcorn, Xwheat, Xcotton;
Positive Variables  xcorn, xwheat, xcotton;
positive variables  Xcorn Xwheat , Xcotton;

In the GAMS formulation of the first example, we have introduced the variable Z in addition to the three variables that featured in the mathematical formulation. Note that GAMS requires the optimization model

Maximize \( cx \)

to have the following form:

Maximize \( z \)

\[ z = cx \]

Here \( z \) is a variable, also called objective variable. Observe that it is the objective variable that is maximized, not the function \( cx \). The name of the objective variable may be chosen freely by the user, like any other variable name. The objective variable is a free variable, which means that it has no bounds. Hence, it each optimization problem there must always be at least one free variable.

Given the requirement of an objective variable in an optimization problem, we need to declare a new free variable and introduce an equation for it. In our first example we declared \( Z \) as a free variable, then we declared and specified the equation \( \text{obj} \) setting \( Z \) equal to the objective function expression and we instructed the solver to maximize \( Z \). The relevant lines of code from the first example follow:

Variables  Z;
Equation  \( \text{obj} \), \( \text{land} \), \( \text{labor} \);
obj..  \( Z =e= 109 \times Xcorn + 90 \times Xwheat + 115 \times Xcotton; \)
solve farmproblem using LP maximizing Z;

Note
In some optimization problems users do not need to introduce a new variable to serve as an objective variable, but a free variable that already features in the model may be used. For example, in the second example above, \( ba \) is a free variable that is used as objective variable, since the model type NLP requires an objective variable. We will discuss this further below.
4.12.2.2 Equations in the Example Models

Each equation in a model must first be declared with an equation declaration statement and then defined with an equation definition statement. We repeat the respective statement from the three examples below:

- First example:
  
  Equations obj, land, labor;
  
  obj.. Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
  land.. Xcorn + Xwheat + Xcotton =l= 100;
  labor.. 6 * Xcorn + 4 * Xwheat + 8 * Xcotton =l= 500;

- Second example:
  
  Equations Pdemand, Psupply, Equilibrium;
  
  Pdemand.. P =g= 6 - 0.3*Qd;
  Psupply.. ( 1 + 0.2*Qs) =g= P;
  Equilibrium.. Qs =g= Qd;

- Third example:
  
  Equations r1, r2, r3, r4, b1, b2;
  
  r1.. ba * so4 =e= 1 ;
  r2.. baoh / ba / oh =e= 4.8 ;
  r3.. hso4 / so4 / h =e= .98 ;
  r4.. h * oh =e= 1 ;
  b1.. ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
  b2.. 2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;

Note that the equation declaration statement begins with the keyword Equation(s). Its main purpose is to name the equation(s). For further details on equation declarations, see section Declaring Equations.

The algebraic structure of the equation is specified in the equation definition statement, where the name of the equation is followed by two dots .. and the algebra. The relation between the left-hand side and right-hand side is specified by special symbols indicating the equation type. The most common symbols are =e= for equality, =l= for a less than or equal to relation and =g= for a greater than or equal to relation. An overview of all GAMS equation types is given in Table Equation Types. For more information on equation definitions, see sections Defining Equations and Conditional Equations.

4.12.2.3 Model Definitions in the Example Models

Once all structural elements of a model have been defined and specified, a model statement is used to define the model that will be solved. The model statement form the three examples follow:

- First example:
  
  Model farmproblem / obj, land, labor /;

- Second example:
  
  Model problem / Pdemand.Qd, Psupply.Qs, Equilibrium.P /;
• Third example:

Model wall / all / ;

Note that a model statement always starts with the keyword model. It is followed by the name of the model which users may choose freely and a list of the equations that are part of the model. Observe that the list of equations in the first example contains all equations that were declared and defined. However, this need not be the case. It is possible to declare and define an equation and then not use it in a model. For example, users could define several models that contain different subsets of equations. If all equations that were previously declared and defined enter the model, the keyword all may be used as a shortcut, like in the third example.

Attention

Only equations that have been declared may be listed in a model statement. All equations in a model must have been defined before the model can be solved.

Note that equilibrium problems like the second example are solved as Mixed Complementarity Problems (MCPs). MCPs require a special variant of the model statement where not only the equations are listed, but also their complementarity relationships. The complementarity relationship between an equation and its associated variable is marked with a dot ".". Hence, in the second example, the equation demand is perpendicular to the variable Qd, the equation supply is perpendicular to the variable Qs and the equation Equilibrium is perpendicular to the variable P.

In the three model statements above the name of the model was immediately followed by the list of the equations included in the model. Observe that an optional explanatory text may be inserted after the name of the model.

4.12.2.4 Solve Statements in the Example Models

After the model has been defined it remains to be solved, i.e. to find a solution for the variables. The solve statement directs GAMS to use a solver to optimize the model or solve the system of equations. The solve statements of the three examples follow:

• First example:

    solve farmproblem using LP maximizing Z;

• Second example:

    solve problem using MCP;

• Third example:

    solve wall using nlp minimizing ba;

Note that a solve statement always begins with the keyword solve followed by the name of the model, as previously specified with a model statement, the keyword using and the type of the model. We have an LP (linear programming problem) in the first example, an MCP (mixed complementarity problem) in the second example and an NLP (nonlinear programming program) in the third example. A complete list of GAMS model types is given in section Classification of Models. After the model type the direction of the optimization is given: either maximizing as in the first example or minimizing as in the third example. In the final position is the name of the objective variable that is to be optimized.
MCPs and systems of equations of the model type Constrained Nonlinear System (CNS) do not have objective variables. Therefore their solve statements end with the model type.

Observe that the system of nonlinear equations in the second example was expressed as an NLP that requires an objective function and a related objective variable. Actually this is an older practice in GAMS as the GAMS model type CNS was added after this example was initially formulated. Hence, we could alternatively use the following solve statement:

```gams
solve wall using cns;
```

However, note that the model type CNS can only be solved by select solvers and cannot incorporate integer variables. A formulation as an optimization problem relaxes these restrictions, thus allowing the use of the model type MINLP and all NLP solvers. If the model type NLP or one of its variants is chosen, one of the variables must be selected as objective variable to be optimized. Observe that which variable is chosen may not really have any effect since a feasible solution requires all of the simultaneous equations to be solved. Thus, while in the third example the variable `ba` is maximized, there is no inherent interest in attaining its maximum - it is just convenient.

### 4.12.2.5 Starting Points

In the third example we have the following line before the solve statement:

```gams
ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;
```

Note that this line provides starting points for the variables in the model. The suffix `.l` appended to a variable name denotes the level value of that variable. If a level value of a variable is specified before the solve statement, this value will act as a starting point for the search algorithms. The level value of a variable is one example of a variable attribute. Other examples include lower and upper bounds which may be set with the suffixes `.lo` and `.up`. For more information on variable attributes, see section Variable Attributes. For guidance on how to choose good starting points, see the tutorial Good NLP Formulations. Specifying starting points may be important for avoiding numerical problems in the model solution. For more on this topic, see the tutorial Finding and Fixing Execution Errors and Performance Problems.

Observe that the statements above are the first examples of assignment statements in this tutorial. Assignment statements play a crucial role in GAMS, they are introduced in section The Assignment Statement.

### 4.12.3 Running a GAMS Job

The model formulation of a GAMS model is saved in a text file with the extension `.gms`, say `myfile.gms`. Then the file is submitted to GAMS. GAMS will execute the instructions in the `.gms` file, with the result that calculations are done, solvers are used and an output file with the solution results is created. The output file is also called the listing file. By default, the name listing file of the input file `myfile.gms` will be `myfile.lst`.

There are two ways to submit a job to GAMS: via the command line and via the GAMS IDE.
4.12 Quick Start Tutorial

4.12.3.1 Running GAMS on the Command Line

The model formulation of a GAMS model is saved in a text file with the extension .gms, say myfile.gms. This file may be run with GAMS using the following call:

> gams myfile

Note the extension .gms may be omitted. This basic GAMS call may be extended with arguments that are called command line parameters. The following example serves as illustration:

> gams transport pw=80 ps=9999 s=mysave

Note that there are three command line parameters: the first will set the page width to 80, the second will set the page length to 9999 and the third will have the effect that a work file named mysave is saved. GAMS offers many command line parameters, they are introduced and discussed in detail in chapter The GAMS Call and Command Line Parameters.

4.12.3.2 Running GAMS with the IDE

The GAMS Integrated Development Environment is a graphical user interface for Windows that facilitates editing, development, debugging GAMS models and running GAMS jobs.

4.12.4 Examining The Output

The output of a GAMS run is saved in the output or listing file. The listing file may contain many parts. Some parts may be suppressed by the modeler, other parts are suppressed by default and need to be activated. For an introduction and detailed discussion, see chapter GAMS Output.

In this section we will review the output that is generated by running the three example models.

4.12.4.1 The Echo Print

The listing file always begins with the echo print. The echo print is a copy of the input file with added line numbers. For example, the echo print of the first example is given below:

1 Positive Variables Xcorn, Xwheat, Xcotton;
2 Variables Z;
3
4 Equations obj, land, labor;
5
6 obj.. Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
7 land.. Xcorn + Xwheat + Xcotton =l= 100;
8 labor.. 6 * Xcorn + 4 * Xwheat + 8 * Xcotton =l= 500;
9
10 Model farmproblem / obj, land, labor /;
11
12 solve farmproblem using LP maximizing Z;

Note that the echo print of the other two examples follows the same principles. Note further, that the echo print serves as an important reference guide since GAMS reports the line numbers of any errors that were detected and of solve and display statements.

Observe that usually the code of even experienced GAMS modelers will contain compilation errors. They are marked in the echo print. For more information on compilation error and how to resolve them, see section Compilation Errors and the tutorial Fixing Compilation Errors.
4.12.4.2 Model Generation Output

Once GAMS has successfully compiled the input file and any numerical computations in assignments have been performed, the solve statements will be executed. The first step is generating a computer readable version of the equations in the problem that will be passed on to an appropriate solver system. During the model generation phase GAMS creates the following output:

- A listing of the equations of the model.
- A listing of the variables of the model.
- A summary of the model structure.
- If errors were detected during model generation they will be reported as well.

4.12.4.2.1 The Equation Listing  The equation listing is the first part of the output generated by a solve statement. By default, the first three equations in every block are listed. The equation listing of the first two equations of each of the three examples are given below:

- **First example:**
  
  Equation Listing  SOLVE farmproblem Using LP From line 12
  
  ---- obj  =E=
  
  obj.. - 109*Xcorn - 90*Xwheat - 115*Xcotton + Z =E= 0 ; (LHS = 0)
  
  ---- land  =L=
  
  land.. Xcorn + Xwheat + Xcotton =L= 100 ; (LHS = 0)

- **Second example:**
  
  Equation Listing  SOLVE problem Using MCP From line 10
  
  ---- Pdemand  =G=
  
  Pdemand.. P + 0.3*Qd =G= 6 ; (LHS = 0, INFES = 6 ****)
  
  ---- Psupply  =G=
  
  Psupply.. - P + 0.2*Qs =G= -1 ; (LHS = 0)

- **Third example:**
  
  Equation Listing  SOLVE wall Using NLP From line 16
  
  ---- r1  =E=
  
  r1.. (1)*ba + (1)*so4 =E= 1 ; (LHS = 1)
  
  ---- r2  =E=
  
  r2.. - (1)*ba + (1)*baoh - (1)*oh =E= 4.8 ; (LHS = 1, INFES = 3.8 ****)
4.12 Quick Start Tutorial

Note that the equation listing is a representation of the algebraic structure of the linear terms in the equation and a local representation containing the first derivatives of the nonlinear terms. The nonlinear terms are automatically placed in parentheses to indicate a local approximation. For further details, see section The Equation Listing in chapter GAMS Output.

4.12.4.2.2 The Column Listing  The column or variable listing contains a list of the individual coefficients sorted by column rather than by row (like in the equation listing). By default, the first three entries for each variable are shown, along with their lower bound \( .lo \), upper bound \( .up \) and current level values \( .l \). The column listing of the first two variables for each of the three examples are given below.

- **First example:**
  
  Column Listing  
  
  SOLVE farmproblem Using LP From line 12

  ---- Xcorn

  Xcorn
  
  (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  
  -109  
  1  land
  6  labor

  ---- Xwheat

  Xwheat
  
  (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  
  -90  
  1  land
  4  labor

- **Second example:**

  Column Listing  
  
  SOLVE problem Using MCP From line 10

  ---- P

  P
  
  (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  
  1  Pdemand
  -1  Psupply

  ---- Qd

  Qd
  
  (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  
  0.3  Pdemand
  -1  Equilibrium

- **Third example:**

  Column Listing  
  
  SOLVE wall Using NLP From line 16
4.12.4.2.3 Model Statistics  The final information generated while a model is being prepared for solution, is the statistics block. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of the third example follow.

MODEL STATISTICS

<table>
<thead>
<tr>
<th>BLOCKS OF EQUATIONS</th>
<th>6</th>
<th>SINGLE EQUATIONS</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCKS OF VARIABLES</td>
<td>6</td>
<td>SINGLE VARIABLES</td>
<td>6</td>
</tr>
<tr>
<td>NON ZERO ELEMENTS</td>
<td>20</td>
<td>NON LINEAR N-Z</td>
<td>10</td>
</tr>
<tr>
<td>DERIVATIVE POOL</td>
<td>20</td>
<td>CONSTANT POOL</td>
<td>16</td>
</tr>
<tr>
<td>CODE LENGTH</td>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information on model statistics, see section The Model Statistics in chapter GAMS Output.

4.12.4.3 The Solution Report

The final major component of the listing file is the solution output. It consists of a summary, some solver-specific output and a report of the solutions for equations and variables.

4.12.4.3.1 The Solve Summary  The solve summary is very important since it contains an overview of key information of the solution. The solve summary of the third example is given below.

SOLVE SUMMARY

<table>
<thead>
<tr>
<th>MODEL</th>
<th>wall</th>
<th>OBJECTIVE</th>
<th>ba</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>NLP</td>
<td>DIRECTION</td>
<td>MINIMIZE</td>
</tr>
<tr>
<td>SOLVER</td>
<td>CONOPT</td>
<td>FROM LINE</td>
<td>16</td>
</tr>
</tbody>
</table>

**** SOLVER STATUS  1 Normal Completion
**** MODEL STATUS   2 Locally Optimal
**** OBJECTIVE VALUE 1.0000

RESOURCE USAGE, LIMIT  0.034  1000.000
ITERATION COUNT, LIMIT 7 2000000000
EVALUATION ERRORS 0 0

For details on the solve summary including possible values for SOLVER STATUS and MODEL STATUS, see section The Solve Summary in chapter GAMS Output.
4.12.4.3.2 The Solver Report  The next part of the listing file is the solver report. It contains solver specific output. The respective output of the third example follows:

CONOPT 3 24.7.3 r58181 Released Jul 11, 2016 DEG x86 64bit/MacOS X

C O N O P T 3 version 3.17A
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark

Pre-triangular equations: 0
Post-triangular equations: 0

** Optimal solution. There are no superbasic variables.

For more on solver reports, see section Solver Report in chapter GAMS Output.

4.12.4.3.3 The Solution Listing  The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed including their level and marginal values and their lower and upper bounds. The solution listing of the first example follows:

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>EQU obj</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>----</td>
<td>EQU land</td>
<td>-INF</td>
<td>100.0000</td>
</tr>
<tr>
<td>----</td>
<td>EQU labor</td>
<td>-INF</td>
<td>500.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>VAR Xcorn</td>
<td>.</td>
<td>50.0000</td>
</tr>
<tr>
<td>----</td>
<td>VAR Xwheat</td>
<td>.</td>
<td>50.0000</td>
</tr>
<tr>
<td>----</td>
<td>VAR Xcotton</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>----</td>
<td>VAR Z</td>
<td>-INF</td>
<td>9950.0000</td>
</tr>
</tbody>
</table>

Note that single dots '.' represent zeros. The extended range arithmetic symbols \(-\text{INF}\) and \(+\text{INF}\) denote minus and plus infinity respectively. \(\text{EPS}\) is another extended range arithmetic symbol that often appears in solution listings. In this context, it is used to indicate basis status in the degenerate case. For example, a basic variable is indicated by a zero (single dot) marginal, while a non-basic variable whose marginal is (nearly) zero is indicated by a marginal of \(\text{EPS}\). For further details, see section The Solution Listing in chapter GAMS Output.

4.12.5 Exploiting the Algebraic Structure

GAMS was deliberately designed to make it easy to express algebraic relationships. Thus it provides notation for indexed operations like sums. Assume that \(z\) is defined as the sum over \(x_i\) where \(i = \{1, 2, 3\}:\n
\[
z = \sum_{i=1}^{3} x_i = x_1 + x_2 + x_3
\]

This can be expressed in GAMS in the following way:
Here \( i \) is a set with three elements, \( z \) is a scalar or variable and \( x(i) \) is a parameter or variable defined over the set \( i \). Note that the keyword \texttt{sum} automatically cycles through all elements of the set \( i \).

Indexed operations like \texttt{sum} may be used in the algebraic specification of equations or in the context of assignments where the value of identifiers are computed. In the following two sections we will introduce revised versions of the first two examples, where we illustrate how using \texttt{sum} simplifies the code and enables more general formulations.

### 4.12.5.1 Revised First Example

Recall that the first example is the following LP:

\[
\begin{align*}
\text{max } & \quad 109X_{\text{corn}} + 90X_{\text{wheat}} + 115X_{\text{cotton}} \\
\text{s.t. } & \quad X_{\text{corn}} + X_{\text{wheat}} + X_{\text{cotton}} \leq 100 \quad \text{(land)} \\
& \quad 6X_{\text{corn}} + 4X_{\text{wheat}} + 8X_{\text{cotton}} \leq 500 \quad \text{(labor)} \\
& \quad X_{\text{corn}} \geq 0 \quad \text{(nonnegativity)} \\
& \quad X_{\text{wheat}} \geq 0 \quad \text{(nonnegativity)} \\
& \quad X_{\text{cotton}} \geq 0 \quad \text{(nonnegativity)}
\end{align*}
\]

Note that this is a special case of the general resource allocation problem that can be written as follows:

\[
\begin{align*}
\text{max } & \quad \sum_j c_j X_j \\
\text{s.t. } & \quad \sum_j a_{ij} X_j \leq b_i \quad \text{for all } i \\
& \quad X_j \geq 0 \quad \text{for all } j,
\end{align*}
\]

where

- \( j = \{\text{corn, wheat, cotton}\} \)
- \( i = \{\text{land, labor}\} \)
- \( X_j = \{X_{\text{corn}}, X_{\text{wheat}}, X_{\text{cotton}}\} \)
- \( c_j = \{109, 90, 115\} \)
- \( a_{ij} = \begin{pmatrix} 1 & 1 & 1 \\ 6 & 4 & 8 \end{pmatrix} \)
- \( b_i = \{100, 500\} \).

This model may be formulated in GAMS in the following way:

```gams
Set j / corn, wheat, cotton / i / land, labor /;
Parameter c(j) / corn 109, wheat 90, cotton 115 / b(i) / land 100, labor 500 /;
Table a(i,j)
    corn wheat cotton
land    1      1      1
labor   6      4      8
```

Positive Variables \( x(j) \);
Variables \( \text{profit} \);

Equations
\[
\text{objective} \\
\text{constraint}(i) \\
\]
\[
\text{objective}.. \quad \text{profit} = e = \sum(j, (c(j)) \times x(j)) \\
\text{constraint}(i) .. \quad \sum(j, a(i,j) \times x(j)) = l = b(i) \\
\]

Model resalloc /all/;
solve resalloc using LP maximizing profit;

We will discuss the components of this model in section Components of the Revised Example Models below.

4.12.5.2 Revised Second Example

Recall that the second example is an economic equilibrium model with a single commodity:

- **Demand Price:** \( P \geq P_d = 6 - 0.3Q_d \)
- **Supply Price:** \( P \leq P_s = 1 + 0.2Q_s \)
- **Quantity Equilibrium:** \( Q_s \geq Q_d \)
- **Nonnegativity:** \( P \geq 0 \), \( Q_s \geq 0 \), \( Q_d \geq 0 \)

A more general formulation of this model accommodates multiple commodities. Consider the following formulation where \( c \) denotes the commodities:

- **Demand Price for** \( c \): \( P_c \geq P_{d_c} = I_{d_c} - \sum_{c} S_{d_c, cc} Q_{d cc} \) for all \( c \)
- **Supply Price for** \( c \): \( P \leq P_{s_c} = I_{s_c} + \sum_{c} S_{s_c, cc} Q_{s cc} \) for all \( c \)
- **Quantity Equil. for** \( c \): \( Q_{s c} \geq Q_{d c} \) for all \( c \)
- **Nonnegativity:** \( P_c \geq 0 \) for all \( c \), \( Q_{s c} \geq 0 \) for all \( c \), \( Q_{d c} \geq 0 \) for all \( c \),

where

- \( P_c \) is the price of commodity \( c \),
- \( Q_{d c} \) is the quantity demanded of commodity \( c \),
- \( P_{d c} \) is the price from the inverse demand curve for commodity \( c \),
- \( Q_{s c} \) is the quantity supplied of commodity \( c \),
- \( P_{s c} \) is the price from the inverse supply curve for commodity \( c \),
- \( cc \) is an alternative notation for the commodities and is equivalent to \( c \),
- \( I_{d c} \) is the inverse demand curve intercept for \( c \),
- \( S_{d c, cc} \) is the slope of the inverse demand curve. It is used to model the effect of buying one unit of commodity \( cc \) on the demand price of commodity \( c \). When \( c = cc \) it is an own commodity effect and when \( c \neq cc \) it is a cross commodity effect.
• $I_{S_c}$ is the inverse supply curve intercept for $c$,

• $S_{sc,cc}$ is the slope of the inverse supply curve. It is used to model the effect of supplying one unit of commodity $cc$ on the supply price of commodity $c$. When $c = cc$ it is an own commodity effect and when $c \neq cc$ it is a cross commodity effect.

This model may be formulated in GAMS in the following way:

```gams
Set commodities / corn, wheat /;
Set curvetype / supply, demand/;
Table intercepts(curvetype,commodities)
  corn   wheat
   demand  4    8
   supply  1    2;
Table slopes(curvetype,commodities,commodities)
  corn   wheat
   demand.corn -.3  -.1
   demand.wheat -.07 -.4
   supply.corn  .5   .1
   supply.wheat  .1  .3;
Positive Variables P(commodities)
             Qd(commodities)
             Qs(commodities);
Equations PDemand(commodities)
              PSupply(commodities)
              Equilibrium(commodities);
Alias (cc,commodities);
Pdemand(commodities)..
  P(commodities) =g= intercepts("demand",commodities)
                   + sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
  intercepts("supply",commodities)
       + sum(cc,slopes("supply",commodities,cc)* Qs(cc)) =g= P(commodities);
Equilibrium(commodities)..
  Qs(commodities) =g= Qd(commodities);
Model problem / Pdemand.Qd, Psupply.Qs,Equilibrium.P /;
solve problem using MCP;
```

### 4.12.6 Components of the Revised Example Models

The revised example models have several new features including sets, specific data entry, and variables and equations that are defined over sets. In addition, if the models are run, modelers will notice some differences in the output. In this section we will discuss these new language features and the differences in the output they entail.
4.12 Quick Start Tutorial

4.12.6.1 Sets in the Revised Examples

In the revised examples we used the subscripts i, j, commodities and cc. In GAMS, subscripts are sets. They have to be defined before they may be used as subscripts. Sets are defined with set statements. Consider the set statement from the first revised example:

```
Set  j  / corn, wheat, cotton /
     i  / land, labor /;
```

Note that the statement begins with the keyword set followed by the name of the set and a list of the elements of the set. Note further, that more than one set may be defined with one set statement. In addition, an optional explanatory text may be inserted after the name of the set and also after each set element. For more details on set definitions, see chapter Set Definition.

Observe that the following line appears in the revised second example:

```
Alias (cc,commodities);
```

This is an alias statement that introduces a new alternative name for a set that was defined earlier. In our example, the set commodities was defined in the first line of the code and cc is the alias, the alternative name for the set commodities. Note that more than one alias may be defined in an alias statement. For further information on aliases, see section The Alias Statement: Multiple Names for a Set.

Note that in our example the alias facilitates to consider both the effects of own and cross commodity quantity on the demand and supply price for an item.

4.12.6.2 Data Entry in the Revised Examples

GAMS provides three formats for data entry: scalars, parameters and tables. Usually scalars are defined with a scalar statement, data vectors are defined with a parameter statement and matrices are defined with a table statement.

Note that we used parameter and table statements in the revised examples. In this section we will discuss parameter and table statements. For details on scalar statements, see section Scalars.

Parameters

The parameter format is used to enter items defined over sets. Generally, the parameter format is used for data items that are one-dimensional (vectors), but multi-dimensional data may be entered with a parameter statement as well. Consider the parameter statement from the first revised example:

```
Parameter  c(j) / corn 109, wheat 90, cotton 115 /
         b(i) / land 100, labor 500 /;
```

Note that the statement begins with the keyword parameter followed by the name of the parameter and the set over which the parameter is defined, the index or domain. Then a list follows where a numerical value is assigned to each member of the index set. Note that the referenced elements must have been defined to be members of the respective set. By default, elements of the domain that are not listed in the parameter statement are assigned the value of zero. Note that more than one parameter may be defined with one parameter statement. For further details on parameter statements, see section Parameters.

Tables

The table format is used to enter data that are dependent on two or more sets. Consider the following two table statements from the revised examples:
Table \( a(i,j) \) crop data
\[
\begin{array}{ccc}
\text{corn} & \text{wheat} & \text{cotton} \\
\text{land} & 1 & 1 & 1 \\
\text{labor} & 6 & 4 & 8 \\
\end{array}
\]

Table intercepts(curvetype,commodities)
\[
\begin{array}{cc}
\text{corn} & \text{wheat} \\
\text{demand} & 4 & 8 \\
\text{supply} & 1 & 2 \\
\end{array}
\]

Note that the statement begins with the keyword `table` followed by the name of the table and the sets over which the table is defined. The next line serves as header for the columns of the table, the elements of the set in the second index position are listed here. The elements of the set in the first index position are the headers of the rows. Thus the elements of the two index positions span a grid where numerical values may be entered. Like in the parameter format, the referenced set elements must have been defined to be members of the respective set.

Note
Alignment is crucial in table statements. The numerical entries must be placed in one and only one column of the table.

By default, elements of the domain that are not listed in the table statement are assigned the value of zero. Note that only one table may be defined with a table statement. For further details on table statements, see section `Tables`. Observe that data may also be entered with assignment statements. For more information, see section `Data Entry by Assignment`.

### 4.12.6.3 Indexed Variables and Equations in the Revised Examples

When the algebraic structure of a problem is exploited in modeling, variables and equations are often defined over one or more sets, they are **indexed**. For example, in the first revised example we have the following lines:

Positive Variable \( x(j) \);
Equations constraint(i);

Note that here the variable \( x \) is defined over the set \( j \) and the equation \( \text{constraint} \) is indexed over the set \( i \). Similarly, in the second revised example we have the following variable and equation statements:

Positive Variables \( P(\text{commodities}) \)
\[
\begin{array}{c}
\text{Qd(\text{commodities})} \\
\text{Qs(\text{commodities})} \\
\end{array}
\]
Equations \( \text{PDemand(\text{commodities})} \)
\[
\begin{array}{c}
\text{PSupply(\text{commodities})} \\
\text{Equilibrium(\text{commodities})} \\
\end{array}
\]

Observe that here all positive variables and all equations are indexed over the set \( \text{commodities} \). Such definitions indicate that the variables and equations are potentially defined for every element of the defining set. Thus, for example, a variable \( P \) could exist for each and every element of the set \( \text{commodities} \). However, how many of these potential cases are activated is determined by the respective equation definition statement(s) where the variable \( P \) is used. For further details on indexed variables and equations, see chapter `Variables` and section `Indexed Equations` respectively.

Next, we will discuss the equation definition statements. The respective lines from the first revised example follow:
4.12 Quick Start Tutorial

```
objective.. profit =e= sum(j, (c(j))*x(j));
constraint(i).. sum(j, a(i,j) *x(j)) =l= b(i);
```

Note that the equation `constraint` is indexed over the set `i` and there are no restrictions specified in the equation definition statement. Thus, GAMS will generate a separate equation for every element of the set `i` in the model generation phase.

The same logic applies to the indexed equations of the second revised example whose definition statements are repeated below:

```
Pdemand(commodities)..
P(commodities) =g= intercepts("demand",commodities) + sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
intercepts("supply",commodities) + sum(cc,slopes("supply",commodities,cc)*Qs(cc)) =g= P(commodities);
Equilibrium(commodities)..
Qs(commodities) =g= Qd(commodities);
```

Observe that equations may be defined over only a part of their domain. This restriction is usually achieved with dollar conditions. For details see section Conditional Equations.

4.12.6.4 Differences in the Output

If variables and equations are defined over sets, some parts of the listing file will look different. In particular, there are some changes in the equation listing, the variable listing and the solution listing.

Revised Models: The Equation Listing

Note that indexed variables are given with their indices in the equation listing. In addition, the specific equations generated for each element of the domain are listed under the name of an indexed equation. To illustrate, we present the equation listing of the first revised example below:

```
---- objective =E=
objective.. - 109*x(corn) - 90*x(wheat) - 115*x(cotton) + profit =E= 0 ; (LHS = 0)

---- constraint =L=
constraint(land).. x(corn) + x(wheat) + x(cotton) =L= 100 ; (LHS = 0)
constraint(labor).. 6*x(corn) + 4*x(wheat) + 8*x(cotton) =L= 500 ; (LHS = 0)
```

Revised Models: The Column Listing

Similar to indexed equations in the equation listing, each instance of a variable is listed under the name of an indexed variable in the variable listing. The respective output of the first revised model follows.
Revised Models: The Solution Listing

In the solution listing of the revised models there is a separate line for each element of the index set. The respective output of the first revised model is given below:

--- EQU constraint

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>land</td>
<td>-INF</td>
<td>100.0000</td>
<td>100.0000</td>
</tr>
<tr>
<td>labor</td>
<td>-INF</td>
<td>500.0000</td>
<td>500.0000</td>
</tr>
</tbody>
</table>

--- VAR x

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>corn</td>
<td>.</td>
<td>50.0000</td>
<td>+INF</td>
</tr>
<tr>
<td>wheat</td>
<td>.</td>
<td>50.0000</td>
<td>+INF</td>
</tr>
<tr>
<td>cotton</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
</tr>
</tbody>
</table>

4.12.7 Documenting the GAMS Code

We have now covered the essential GAMS features. However, any good code includes documentation, otherwise it might be useless if it is revisited after a longer time. GAMS offers three ways to document the code: comments, explanatory texts and naming.

Comments

There are several ways to add comments in GAMS. The most common are single line comments and block comments. Single line comments may be inserted on any line by placing an asterisk * in the first column position. The text that follows the asterisk will be completely ignored by the GAMS compiler and may contain any content including GAMS keywords. Note that several successive lines may be single line comments. Block comments are marked with the dollar control option $ontext at the beginning of the comment block and the dollar control option $offtext at the end of the comment. Block comments usually
span several lines, but they may also contain just one line on the one hand and whole sections of the code on the other hand.

In addition, users may freely enter blank lines to set off certain sections of the code and generally enhance readability. For further details on comments, see section Comments.

**Explanatory Text**

Declarations of sets, parameters, variables and equations may include an optional descriptive text that follows the name of the identifier. In addition, the elements of sets may be accompanied by a text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the respective identifier in the output file. Note that the explanatory text may be quoted or unquoted and single or double quotes may be used, but they must match. An example is given below. For further details, see section Text.

**Naming**

Apart from avoiding reserved words, names for identifiers in GAMS may be freely chosen. Some modelers, particularly if they have a background in mathematics, prefer short names like $x(i)$. Other modelers strongly prefer longer descriptive names that makes it easier for them to recall what quantities are denoted. In this case naming is regarded as part of the documentation. GAMS accommodates both styles and modelers may choose which style works best for them. If short names are used, we recommend to generously use descriptive texts and comments for documentation.

To illustrate the virtues of comments, blank lines, explanatory text and long names we will repeat the code of the revised LP from above and then offer an alternative, documented formulation.

```gams
Set       j     / corn, wheat, cotton /    
          i     / land, labor /;
Parameter c(j)  / corn 109, wheat 90, cotton 115 /    
        b(i)  / land 100, labor 500 /;
Table     a(i,j) corn wheat cotton     
 land    1     1     1    
 labor   6     4     8    ;
Positive Variables x(j); Variables profit;
Equations objective constraint(i) ;
objective.. profit =e= sum(j, (c(j))*x(j));
constraint(i).. sum(j, a(i,j) *x(j)) =l=  b(i);
Model resalloc /all/;
solve resalloc using LP maximizing profit;

The documented version with longer names follows:

$ontext
well formatted algebraic version of the first example model
$offtext
Set   products  'Items produced by farm'
```
Parameter netreturns(products) 'Net returns per unit produced'
  / corn 109, wheat 90, cotton 115 /
endowments(resources) 'Amount of each resource available'
  / land 100, labor 500 /

Table resourceusage(resources,products) 'Resource usage per unit produced'
corn wheat cotton
  land  1  1  1
  labor  6  4  8 ;

Positive Variables production(products) 'Number of units produced';
Variables profit 'Total sum of net returns';

Equations profitAcct 'Profit accounting equation'
  available(resources) 'Resource availability limit';

$ontext
  Specify definition of profit
$offtext
profitAcct..
  profit =e= sum(products, netreturns(products)*production(products));

$ontext
  Limit available resources
  Fix at exogenous levels
$offtext
available(resources)..
  sum(products,
    resourceusage(resources,products) *production(products))
=l= endowments(resources);

Model resalloc /all/;
solve resalloc using LP maximizing profit;

4.12.8 Guidelines on Ordering GAMS Statements and Formatting GAMS Programs

A GAMS program is a collection of GAMS statements. In this section we will offer some general guidelines on ordering GAMS statements and formatting GAMS programs.

- GAMS is case insensitive. This applies to GAMS keywords as well as to user-defined names. For example, the keyword VARIABLE is identical to Variable and variable and the user-defined name mincost is identical to minCost and minCOST. However, note that the spelling in the output file is determined by the spelling in the first occurrence of an identifier or label.
Individual GAMS statements may be formatted in almost any style. Multiple lines may be used for a statement, blank lines may be embedded and any number of spaces or tabs may be inserted. In addition, several statements may be placed on one line. Note that they have to be separated by semicolons ;.

We recommend that every GAMS statement is terminated with a semicolon ;. However, note that semicolons are not mandatory if the next word is a GAMS keyword.

Identifiers like sets, parameters, scalars, tables, acronyms, variables, equations, models and files must first be declared and defined before they may be used in the code. An identifier is named in a declaration statement and specific values are assigned to it when it is defined. Often identifiers are defined at the same time they are declared. Note that for equations the declaration and definition statements are always distinct.

The names for identifiers and labels and the explanatory text must follow certain rules. See chapter GAMS Programs and the tutorial Good Coding Practices for more information.

Statements must be ordered such that identifiers are declared before they are used. If identifiers are used on the right-hand side of an assignment statement, they must also have been defined. If they are used in an equation and the equation is included in a model, then they must be assigned data before a solve statement for the respective model. Note that compilation errors will be triggered if this order is not followed.

4.12.9 Adding Complexity

There are several GAMS features that are widely used and serve to add subtlety and complexity to models. They include exception handling through conditionals, displaying data in the output file and report writing where the information from the optimal solution is used to create reports that meet the needs of modelers and their clients.

4.12.9.1 Conditionals

Assignments are often valid only for certain cases and sometimes equations should reasonably be restricted to a subset of the domain over which they were defined. It is easy to model such assignments and equations in GAMS with conditionals. Conditional expressions in GAMS are introduced and discussed in detail in chapter Conditional Expressions, Assignments and Equations. In this section we will only give a quick overview to demonstrate the capabilities of conditionals in GAMS.

Note that the dollar condition is at the heart of conditionals in GAMS. The general form of a conditional expression is as follows:

\[
\text{term \$ logical\_condition}
\]

This translates to: "Do something with 'term' only if the logical condition is true." Observe that term may be a number, a set, a parameter, a variable or an equation. The dollar operator \$ is a GAMS speciality and one of the most powerful features of GAMS. The logical condition may take many different forms, see section Logical Conditions for details.

The following examples illustrate how conditionals in GAMS may be used.

Conditional Assignments

Consider the following example:
$x \ (y > 0) = 10;

Note that $x$ is assigned the value of ten only if the scalar $y$ is greater than zero, otherwise no assignment is made and $x$ keeps its previous value. For more information, see section Conditional Assignments.

**Conditional Indexed Operations**

Consider the following example:

$$z = \text{sum}[i \ (y(i) \neq 0), x(i)];$$

Note that the term $x(i)$ will only be included in the sum if $y(i)$ is nonzero. For further information, see section Conditional Indexed Operations.

**Conditionals in the Domain of Definition of Equations**

Consider the following equation definition statements:

$$\text{Eq1} \ (qq > 0) \ldots \ x\var =e= 3;$$
$$\text{Eq2} \ (\text{sum}(i, q(i)) > 0) \ldots \ y\var =l= 4;$$
$$\text{Eq3}(i) \ (a(i) > 0) \ldots \ ivar(i) =g= -a(i);$$

Note that in each equation the domain of definition is restricted to those cases where the logical condition evaluates to TRUE. For further details, see section Dollar Control over the Domain of Definition.

**Conditionals in the Algebraic Formulation of Equations**

Our last example illustrates how a dollar condition may be used in the body of an equation:

$$\text{Eq4} \ldots \ x\var + y\var \ (qq > 0) =e= 3;$$

Note that the term $y\var$ is included in the equation only if $qq$ is greater than zero, otherwise $y\var$ is treated as if it were zero. For more information, see section Dollar Operators within the Algebra of Equations.

### 4.12.9.2 Displaying Data

In GAMS, the display statement is a quick way to write data into the output file. For example, assume we add the following statement after the solve statement in the revised and documented version of the farm linear programming model above:

`display profit.l, production.l;`

Recall that profit and production are variables. The suffix .l indicates that we wish to display the variable attribute level. The following output will be generated in the listing file:

```
---- 47 VARIABLE profit.L = 9950.000 Total sum of net returns
---- 47 VARIABLE production.L Number of units produced
    corn 50.000, wheat 50.000
```

Observe that the name of the variable, the explanatory text and the respective numerical values are given. In addition to data like parameters, sets, variable attributes, equation attributes and model attributes, quoted text may be displayed. Note that numerical entries that equal zero will not be displayed. For a more detailed introduction, see chapter The Display Statement.
4.12.9.3 Report Writing

In many cases modelers wish to summarize the most important results of the solution in a table for a quick overview. GAMS allows post-solution computations where information from the solution may be used to assign values to new parameters that are then displayed. The information from the solution most often used for report writing includes the level values of variables and equations and the marginal values of equations (also called dual values or shadow prices). Note that variable and equation attributes are accessed as follows:

\[ \text{var_name.sfx} \]
\[ \text{eqn_name.sfx} \]

Here \text{var_name} and \text{eqn_name} is the name of the variable and equation in GAMS respectively. The attribute is denoted by .sfx, where sfx may be l for level and m for marginal. Note that the suffix sfx may take other values as well. For details see sections Variable Attributes and Equation Attributes.

Observe that the numerical values of the levels and marginals of variables and equations are generally undefined until a solve statement is executed. After GAMS has retrieved the solution from the solver, the respective values from the solution are assigned to the attributes. These values remain unchanged until the next solve, where they are replaced with the values from the most recent solution.

In the remainder of this section we will present two examples to illustrate report writing. Assume we add the following report writing sequence after the solve statement to the revised and documented version of the farm linear programming model above:

```
Set item / Total, "Use by", Marginal /;
Set qitem / Available, Corn, Wheat, Cotton, Value /;
Parameter Thisreport(resources,item,qitem) 'Report on resources';
Thisreport(resources,"Total","Available") = endowments(resources);
Thisreport(resources,"Use by",qitem) =
    sum(products$sameas(products,qitem),
        resourceusage(resources,products) * production.l(products));
Thisreport(resources,"Marginal","Value") =
    available.m(resources);

option thisreport:2:1:2;
display thisreport;
```

Note that both, equation marginals (available.m(resources)) and variable levels (production.l(products)) are included in the calculations. The function sameas in the logical condition above returns the value TRUE if the element of the set products is equivalent to the element of the set qitem and FALSE otherwise. Thus this condition ensures that the third index of the parameter Thisreport is identical to the element of the set products in the sum. Observe that with the option statement in the penultimate line the appearance of the display is customized. For details see section Local Display Control. The following report will be generated by the display statement:

```
---- 61 PARAMETER Thisreport Report on resources

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Use by</th>
<th>Use by</th>
<th>Marginal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Available</td>
<td>corn</td>
<td>wheat</td>
<td>Value</td>
</tr>
<tr>
<td>land</td>
<td>100.00</td>
<td>50.00</td>
<td>50.00</td>
<td>52.00</td>
</tr>
<tr>
<td>labor</td>
<td>500.00</td>
<td>300.00</td>
<td>200.00</td>
<td>9.50</td>
</tr>
</tbody>
</table>
```
Similarly, we could add the following report writing sequence to the revised version of the equilibrium model above:

```plaintext
Set qitem / Demand, Supply, "Market Clearing" /;
Set item / Quantity, Price /;

Parameter myreport(qitem,item,commodities);
myreport("Demand","Quantity",commodities) = Qd.l(commodities);
myreport("Supply","Quantity",commodities) = Qs.l(commodities);
myreport("Market Clearing","Price",commodities) = P.l(commodities);

display myreport;
```

Note that in the new parameter the level values for supply and demand as well as the market clearing price are saved. The resulting report follows:

```
---- 39 PARAMETER myreport

<table>
<thead>
<tr>
<th></th>
<th>Corn</th>
<th>Wheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply</td>
<td>Quantity</td>
<td>1.711</td>
</tr>
<tr>
<td>Demand</td>
<td>Quantity</td>
<td>1.711</td>
</tr>
<tr>
<td>Market Clearing</td>
<td>Price</td>
<td>2.671</td>
</tr>
</tbody>
</table>
```

For more on report writing, see chapters The Display Statement and The Put Writing Facility.

### 4.12.10 Advantages of Algebraic Modeling in General

We will conclude this tutorial with a discussion of the advantages of using algebraic modeling in general.

Algebraic modeling languages like GAMS facilitate model formulations in general algebraic terms, that are very concise and readable. Language elements that are essential include sets that may serve as indices, algebraic expressions, indexed operations, powerful sparse index and data handling variables and constraints with user-defined names. Model formulations are largely independent of the data and exact application contexts. Such formulations may be easily transferred to different contexts, data may be added without the need to reformulate the model and the model may be extended to reflect additional complexity.

However, GAMS algebraic requirements and the summation notation are difficult for some users. Some modelers will always prefer the exact problem context, not an abstract general formulation. This may lead to a strategy most modelers use: Start with a small concrete formulations that capture the essence of the problem and support the development of more general GAMS models.

#### 4.12.10.1 One Model - Different Contexts

In the linear programming problem above we modeled profit maximizing in a farm. This model may easily be transferred to another context as follows:
Set products 'Items produced'
   / Chairs, Tables, Dressers /
resources 'Resources limiting production'
   / RawWood, Labor, WarehouseSpace /

Parameter Netreturns(products) 'Net returns per unit produced'
   / Chairs 19, Tables 50, Dressers 75 /
Endowments(resources) 'Amount of each resource available'
   / RawWood 700, Labor 1000, WarehouseSpace 240 /

Table Resourceusage(resources,products) 'Resource usage per unit produced'
  Chairs Tables Dressers
  RawWood  8   20  32
  Labor    12  32  45
  WarehouseSpace  4   12  10

Positive Variables Production(products) 'Number of units produced';
Variables Profit 'Total sum of net returns';

Equations ProfitAcct 'Profit accounting equation'
   Available(resources) 'Resource availability limit';

ProfitAcct..
   Profit =e= sum(products, netreturns(products) * production(products)) ;
Available(resources).. 
   sum(products, resourceusage(resources,products) * production(products)) 
   =l= endowments(resources);

Model resalloc /all/;
solve resalloc using LP maximizing Profit;

Note that in this model we have chairs, tables and dressers instead of corn, wheat and cotton and raw wood, labor and warehouse space instead of land and labor, but the algebraic structure of the model is the same. Thus we still have sets for products and resources, parameters for net returns per unit produced and available resources, a table for resource usage per unit produced and exactly the same variables and equations, the same model and solve statement. Hence, if the algebraic structure for a type of problem is built, it may be used in another context of the same problem type with just minor modifications in the data.

4.12.10.2 Adding More Data

It is easy to add more data to a model. For example, we could add two new products and two new resources to the carpenter model above in the following way:

Set products 'Items produced'
   / Chairs, Tables, Dressers, HeadBoards, Cabinets /
resources 'Resources limiting production'
   / RawWood, Labor, WarehouseSpace, Hardware, ShopTime /

Parameter Netreturns(products) 'Net returns per unit produced'
   / Chairs 19, Tables 50, Dressers 75, HeadBoards 28, Cabinets 25 /
Endowments(resources) 'Amount of each resource available'
User's Guide

/ RawWood 700, Labor 1000, WarehouseSpace 240, Hardware 100, Shoptime 600 /

Table Resourceusage(resources,products) 'Resource usage per unit produced'

<table>
<thead>
<tr>
<th>Chairs</th>
<th>Tables</th>
<th>Dressers</th>
<th>HeadBoards</th>
<th>Cabinets</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawWood</td>
<td>8</td>
<td>20</td>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>Labor</td>
<td>12</td>
<td>32</td>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>WarehouseSpace</td>
<td>4</td>
<td>12</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Hardware</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Shoptime</td>
<td>6</td>
<td>8</td>
<td>30</td>
<td>5</td>
</tr>
</tbody>
</table>

Positive Variables Production(products) 'Number of units produced';
Variables Profit 'Total sum of net returns';

Equations ProfitAcct 'Profit accounting equation'
Available(resources) 'Resource availability limit';

ProfitAcct..
Profit =e= sum(products, netreturns(products) * production(products)) ;

Available(resources)..
sum(products, resourceusage(resources,products) * production(products)) =l= endowments(resources);

Model resalloc /all/;
solve resalloc using LP maximizing Profit;

Observe that the elements HeadBoards and Cabinets were added to the set Products and the elements Hardware and ShopTime were added to the set Resources. In addition, the data in the two parameters and the table was updated to reflect these new labels. However, the model structure remained unchanged. Thus, GAMS models may easily be extended from smaller to larger data sets. Note that this feature may be exploited for model development. Users may develop a model with a small data set and test and debug it. Afterwards, they may move to the full problem data set without having to alter the algebraic structure of the model. For more details on this strategy, see section Small to Large: Aid in Development and Debugging.

4.12.10.3 Extending the Model

Assume we wish to make a model more complex by adding new features. For example, we could extend the carpenter model above to reflect the possibility of renting or hiring additional resources subject to a maximum limit. Consider the following code:

Set products 'Items produced'
    / Chairs, Tables, Dressers /
resources 'Resources limiting production'
    / RawWood, Labor, WarehouseSpace/
hireterms 'Resource hiring terms'
    / Cost, Maxavailable /;

Parameter Netreturns(products) 'Net returns per unit produced'
    / Chairs 19, Tables 50, Dressers 75 /
Endowments(resources) 'Amount of each resource available'
    / RawWood 700, Labor 1000, WarehouseSpace 240 /;
Table Resourceusage(resources,products) 'Resource usage per unit produced'

<table>
<thead>
<tr>
<th>Chairs</th>
<th>Tables</th>
<th>Dressers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawWood</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Labor</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>WarehouseSpace</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

Table Hiredata(resources,hireterms) 'Resource hiring data'

<table>
<thead>
<tr>
<th>Cost</th>
<th>Maxavailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawWood</td>
<td>3</td>
</tr>
<tr>
<td>Labor</td>
<td>12</td>
</tr>
<tr>
<td>WarehouseSpace</td>
<td>4</td>
</tr>
</tbody>
</table>

Positive Variables Production(products) 'Number of units produced'

Variables HireResource(resources) 'Resources hired';

Variables Profit 'Total sum of net returns';

Equations ProfitAcct 'Profit accounting equation'

Available(resources) 'Resource availability limit'

Hirelimit(resources) 'Resource hiring limit';

ProfitAcct.. Profit
  =e= sum(products, Netreturns(products) * Production(products))
  - sum(resources, Hiredata(resources,"cost") * HireResource(resources));

Available(resources)..
  sum(products,
    Resourceusage(resources,products) * Production(products))
=1= Endowments(resources) + HireResource(resources);

Hirelimit(resources)..
  HireResource(resources) =l= Hiredata(resources,"Maxavailable");

Model resalloc /all/;

solve resalloc using LP maximizing Profit;

Observe that we introduced the set hireterms, the table Hiredata, the positive variable HireResource and the equation Hirelimit. In addition, we included new terms in the equations Profit and Available to reflect that through hiring the resources are increased, but hiring comes with a cost diminishing the profit. Thus the algebraic structure of the earlier model could be used as the core for this model that has additional features.

Note that this method may also be exploited for model development. Users may adapt models from other studies customizing them for the problem at hand and thus speeding up the development process. In addition to adapting models from related earlier studies that were done by the modeler or his group, model development may be jumpstarted by adapting models from the extensive GAMS Model Library.

4.13 Good Coding Practices

The GAMS language is quite flexible regarding the syntax and format of the code it accepts, offering users considerable latitude in how they organize and format their GAMS code. Most modelers develop their own style as they gain experience with the GAMS system. This tutorial reflects the coding preferences of Bruce A. McCarl (currently professor of Agricultural Economics at Texas A&M University). Note that Bruce has extensive experience with GAMS, both as a modeler and an educator, and many GAMS users
know, use, and benefit from his work. The goal of this tutorial is not to present a rigid set of rules to follow arbitrarily, but rather to help users develop their own coding preferences and style. The larger goal is to build self-documenting models that are easy to read and understand, to edit, and to debug: both for the developer working in the present, and for a larger group of colleagues and consultants working with the model over a span of months or years.

We will cover the following topics:

- Using Longer Names and Descriptive Text
- Including Comments
- Choosing Raw Data Instead Of Computed Data
- Avoiding the Universal Set in the Context of Data Input
- Defining Sets and Subsets Wisely
- Structuring and Formatting Files to Improve Readability
- Other Suggestions

4.13.1 Using Longer Names and Descriptive Text

The readability of GAMS code may be significantly improved by using longer self-explanatory names for identifiers (e.g. names of sets, parameters, variables, etc). Consider the following lines of code from the production and inventory model [ROBERT]:

Sets p 'products' / low, medium, high /
    r 'raw materials' / scrap, new /
    tt 'long horizon' / 1*4 /
    t(tt) 'short horizon' / 1*3 /;

Table a(r,p) input 'coefficients'

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>medium</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrap</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>new</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table c(p,t) 'expected profits'

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>25</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>medium</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>high</td>
<td>75</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

Variables x(p,tt) 'production and sales'
    s(r,tt) 'opening stocks'
    profit;

Positive variables x, s;

Equations cc(t) 'capacity constraint'
    sb(r,tt) 'stock balance'
    pd 'profit definition' ;

cc(t).. sum(p, x(p,t)) =l= m;
sb(r,tt+1) =e= s(r,tt) - sum(p, a(r,p)*x(p,tt));

pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
  - sum(r, misc("storage-c",r)*s(r,t)))
  + sum(r, misc("res-value",r)*s(r,"4"));

s.up(r,"1") = misc("max-stock",r);

These lines may be reformatted in the following way (see (good.gms)):

Sets process /low 'uses a low amount of new materials',
    medium 'uses a medium amount of new materials',
    high 'uses a high amount of new materials'/
rawmaterial /scrap, new/
Quarters /spring, summer, fall, winter/
quarter(Quarters) /spring, summer, fall/
Table usage(rawmaterial,process) 'input coefficients'
    low    medium  high
scrap  5        3       1
new    1        2       3
Table expectprof(process,quarters) 'expected profits'
    spring summer fall
low     25       20      10
medium  50       50      50
high    75       80      100;
Variables production(process,Quarters) 'production and sales'
    openstock(rawmaterial,Quarters) 'opening stocks'
    profit;
Positive variables production, openstock;
Equations capacity(quarter) 'capacity constraint'
    stockbalan(rawmaterial,Quarters) 'stock balance'
    profitacct 'profit definition';

capacity(quarter)..
    sum(process, production(process,quarter)) =l= mxcapacity;
stockbalan(rawmaterial,Quarters+1)..
    openstock(rawmaterial,Quarters+1) =e=
    openstock(rawmaterial,Quarters)
    - sum(process, usage(rawmaterial,process)
        *production(process,Quarters));
profitacct.. profit =e=
    sum(quarter,
        sum(process, expectprof(process,quarter)
            *production(process,quarter))
    - sum(rawmaterial, miscdata("store-cost",rawmaterial)*
        openstock(rawmaterial,quarter)))
    + sum(rawmaterial, miscdata("endinv-value",rawmaterial)
        *openstock(rawmaterial,"winter"));
openstock.up(rawmateral,"spring") = miscdata("max-stock",rawmateral);

Note that the two formulations are equivalent in their effect, but in the second formulation longer, more descriptive names were used for the sets, tables, variables and equations. In addition, longer names were used for the set elements and in the definition of the set process the set elements have additional explanatory text. Observe that the second formulation is easier to understand. This will be particularly useful if and when the code is revisited in 5 years' time.

Note
- Recall that GAMS allows long names for identifiers and labels (set elements). Users may exploit this feature to introduce long descriptive names. However, note that names for labels that are longer than 10 characters do not work well in multi-column displays. See the paragraph on customizing display width for details.
- Use explanatory text for identifiers to indicate units, sources, descriptions, etc. It's not that hard to do and it pays dividends later.
- Similarly, use explanatory text for set elements as appropriate.

For example, the descriptive text in the in the second line in the following code snippet is much more informative than the text in the first line:

Parameter vehsales(r) 'regional vehicle sales';
Parameter vehsales(r) 'regional vehicle sales ($ millions/yr)';

Note that the descriptive text will be displayed whenever the respective identifier is displayed. Hence, including units in the text will save time if results will have to be interpreted later.

4.13.2 Including Comments on Procedures and the Nature and Sources of Data

We recommend that the documentation of the code offers answers to the following questions:

- What are the units of the variables and parameters?
- Where did the data come from?
- What are the characteristics of the data such as units and year of applicability?
- Why was a constraint set up in the way it is implemented?

In addition, it is often helpful to add comments that describe assumptions, the intent of equation terms, data sources, including document name, page number, table number, year of applicability, units, URL etc.

Consider the following example where various forms of comments are illustrated:
4.13 Good Coding Practices

* this is a one line comment that could describe data

$ontext
My data could be described in this multi-line comment
This is the second line
$offtext

* The following dollar control option activates end-of-line comments
* and redefines the symbol for end-of-line comments

$eolCom #
x = sum(i, z(i)) ; # this is an end-of-line comment

* The following dollar control option activates in-line comments
* and redefines the symbols for in-line comments

$inLineCom (* *)
x = sum(i, z(i)) ; (* this is an in-line comment *) r = sum(i, z(i)) ;

For more information on comments in GAMS, see section Comments.

4.13.3 Choosing Raw Data Instead Of Computed Data

Modelers often have a choice how they enter data: they could either use raw data and transform it to the extent needed inside GAMS or process data externally and enter the final results into GAMS. The second choice may be attractive if the raw data is available in a spreadsheet where it can be manipulated before it is introduced to GAMS. However, over time spreadsheets and other data manipulation programs change or get lost and often these programs are not documented well. Therefore, we recommend to enter data into GAMS in a form that is as close as possible to the actual collected data and then manipulate the data with GAMS to obtain the desired form. This will make it much easier to update models later or to work out implicit assumptions.

4.13.4 Avoiding the Universal Set in the Context of Data Input

While GAMS permits using the universal set * as an index in a parameter or table statement, in most cases it is not advisable to do so. Consider the following example from the production and inventory model [ROBERT]:

Sets r 'raw materials' / scrap, new /;
Table misc(*,r) 'other data'
  scrap new
  max-stock 400 275
  storage-c .5 2
  res-value 15 25;

pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
  - sum(r, misc("storage-c",r)*s(r,t)))
  + sum(r, misc("res-value",r)*s(r,"4"));

Note that the definition of the table misc indicates that any entry in the first index position is allowed. There is no domain checking. Consequently, if the label res-value is misspelled as res-val in the equation pd, GAMS will compile and execute the program without signaling an error, but instead of the expected values (i.e. misc(r,"res-value")), the values of misc(r,"res-val") will be used in the equation. These zero values will lead to faulty results, and the modeler will not be alerted to this fact. To ensure that the results of a GAMS run are reliable and trustworthy, we strongly recommend to use domain checking by introducing a new set for the labels in the first index position of the table misc:
Sets  
   r   'raw materials'
    / scrap, new /
  miscitem 'misc input items'
    / max-stock, storage-c, res-value /;

Table  misc(miscitem,r) 'other data'

<table>
<thead>
<tr>
<th></th>
<th>scrap</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-stock</td>
<td>400</td>
<td>275</td>
</tr>
<tr>
<td>storage-c</td>
<td>.5</td>
<td>2</td>
</tr>
<tr>
<td>res-value</td>
<td>15</td>
<td>25</td>
</tr>
</tbody>
</table>

Observe that the new set miscitem contains exactly the labels that appear in the rows of the table misc. Hence the set miscitem may be used in the first index position of the definition of misc without loss of generality, but with the benefit of domain checking.

### 4.13.5 Defining Sets and Subsets Wisely

Generally, the elements of a set have a feature in common or they are similar in some way. In this section we will give some guidance on how to partition the labels in the data into sets. In addition, we will discuss in which contexts it is useful to introduce subsets. For an introduction to sets in GAMS, see chapter Set Definition.

For example, suppose we have three grades of oil and three processes to crack it. The question arises whether we should introduce one set with nine elements or two sets with three elements and a two-dimensional set. We recommend the second alternative.

In another example, we consider a budget for farm spending: we have annual (i.e. cumulative yearly) spending for fertilizer and for seed and also monthly spending for labor and for water. There are 26 decisions or items in the budget. We could introduce a set with 26 elements or we could use the following formulation:

```gams
Sets resources / fertilizer, seed, labor, water /
   periods  / jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, annual /
   use(resources,periods)
    / (fertilizer,seed).annual
       (labor,water)  .(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec) /;
```

We recommend the formulation above and to err on the side of being more extensive or exact with set definitions.

Occasionally it is necessary to group some labels into one set for a certain purpose and then single out some of them for another purpose. Subsets facilitate modeling such a case. For example, a set of all cities in a model may be needed to enter distances and compute related transportation costs. In addition, a subset can be used to specify the cities that are hubs for some activity, since some equations should be restricted to these hubs.
4.13.6 Structuring and Formatting Files to Improve Readability

In this section we will offer some guidelines on structuring and formatting the GAMS code to make it easy to read.

There are several ways to structure the GAMS code. Two styles are outlined in section Organization of GAMS Programs. The following recommendation to enter the sections of the code in a fixed order is an extended version of the first style:

1. Set definitions for sets that are data related
2. Parameter, scalar and table definitions, possibly intermixed with calculations
3. Variable definitions
4. Equation declarations
5. Equation definitions (algebraic specification of equations)
6. Model and solve statement(s)
7. Definitions of sets and parameters for report writing
8. Calculations for report writing
9. Display statement(s) for reports

Note that the code will be easiest to navigate if each section of the code contains only one type of statements. For example, interspersing set definitions with parameter definitions will make the code unnecessarily difficult to read.

In addition to following a fixed structure, it is also essential to properly format the code. Of course, formatting is in many respects a matter of taste. The following list offers some ideas:

- Align the names of identifiers and descriptive text, as demonstrated in the examples in this tutorial and in the GAMS User's Guide in general.
- Use spacing and indents.
- Use blank lines to highlight something and to mark sections of the code.
- Ensure that variables and all their index positions are on one line in equation definitions.
- Indent in indexed operations like sums and programming flow control structures like loops and if statements to delineate terms. The structure of a long and complex statement may be revealed through careful indentation and thoughtful placement of closing parentheses.

We will demonstrate the effect of proper formatting with the following two examples. The first example contains valid GAMS code, but is deliberately poorly formatted:

```gams
Sets    products 'available production process' / low 'uses low new materials'
         medium 'uses medium new materials', high 'uses high new materials'
         rawmaterial 'source of raw materials' / scrap, new /
         Quarters 'long horizon' / spring, summer, fall, winter /
         quarter(Quarters) 'short horizon' / spring, summer, fall /
 Variables production(products,Quarters) 'production and sales'
         openstock(rawmaterial,Quarters) 'opening stocks', profit
 Positive variables production, openstock;
 Equations capacity(quarter) 'capacity constraint',
     stockbalan(rawmaterial,Quarters) 'stock balance',
     profitacct profit definition ;
 profitacct.. profit =e= sum(quarter, sum(products, expectprof(
     products,quarter) *production(products,quarter))-
     sum(rawmaterial,miscdata("store-cost",rawmaterial)*openstock(rawmaterial,
     ,quarter)))+ sum(rawmaterial, miscdata("endinv-value",rawmaterial) *openstock(rawmaterial,"winter"))
```

The second example is well formatted:

```gams
Sets    products 'available production process' / low 'uses low new materials'
         medium 'uses medium new materials', high 'uses high new materials'
         rawmaterial 'source of raw materials' / scrap, new /
         Quarters 'long horizon' / spring, summer, fall, winter /
         quarter(Quarters) 'short horizon' / spring, summer, fall /
 Variables production(products,Quarters) 'production and sales'
         openstock(rawmaterial,Quarters) 'opening stocks', profit
 Positive variables production, openstock;
 Equations capacity(quarter) 'capacity constraint',
     stockbalan(rawmaterial,Quarters) 'stock balance',
     profitacct profit definition ;
 profitacct.. profit =e= sum(quarter, sum(products, expectprof(
     products,quarter) *production(products,quarter))-
     sum(rawmaterial,miscdata("store-cost",rawmaterial)*openstock(rawmaterial,
     ,quarter)))+ sum(rawmaterial, miscdata("endinv-value",rawmaterial) *openstock(rawmaterial,"winter")))
```
The second example contains the same code as the first example, but is carefully formatted:

Sets products 'available production process'
   / low 'uses a low amount of new materials',
   medium 'uses a medium amount of new materials',
   high 'uses a high amount of new materials'/
rawmateral 'source of raw materials' / scrap, new /
Quarters 'long horizon' / spring, summer, fall, winter /
quarter(Quarters) 'short horizon' / spring, summer, fall /
;

Variables production(products,Quarters) 'production and sales'
   openstock(rawmateral,Quarters) 'opening stocks'
   profit ;

Positive Variables production, openstock ;

Equations capacity(quarter) 'capacity constraint'
   stockbalan(rawmateral,Quarters) 'stock balance'
   profitacct 'profit definition' ;

profitacct..
   profit =e= 
   sum(quarter,
      sum(products, expectprof(products,quarter)
         *production(products,quarter)
      )
      - sum(rawmateral, miscdata("store-cost",rawmateral)*
         openstock(rawmateral,quarter)
      )
   )
   + sum(rawmateral, miscdata("endinv-value",rawmateral)
      *openstock(rawmateral,"winter")
   )
   ;

Observe that inserting blank lines, aligning the names of identifiers and descriptive text, and indenting and formatting closing parentheses in the sums makes the code much easier to read and understand (both now and in the future) and is well worth adopting as standard practice when writing GAMS code.

4.13.7 Other Suggestions

We will complete this tutorial by offering some other useful suggestions that may help modelers develop their own conventions.

Even though GAMS is case insensitive, it is advisable to establish some convention on the use of upper and lower case letters. For example, Paul N. Leiby (currently at Oak Ridge National Laboratory) uses lower case for texts and comments, and upper case for GAMS reserved words and variable and parameter names. The casing used when an identifier or label is first encountered in a GAMS program is the casing stored by GAMS and used in subsequent outputs like the listing file or a GDX file. Any casing can be used (so nowhere is equivalent to nowhere) but the casing stored is determined by first use.

A similar situation holds for label quoting: the type of quotes stored (if any) are determined by first use.

Note that the dollar control option $onSymList will cause a list of all identifier names to be displayed in the compilation output of the output file. This list may be used to review the spelling and casing of the identifiers as they will appear in output files. Similarly, the dollar control option $onUELList will cause
an ordered list of all labels to be displayed in the compilation output of the output file. This is useful for checking both the case and order of the labels used in the GAMS program. For more on issues related to label ordering, see section Ordered and Unordered Sets.

To keep track of the data types of identifier names, some modelers always start set names with $s$-, names of parameters with $d$-, variables names with $v$- and equation names with $e$-.

Some experienced GAMS users always surround explanatory text with quotes: this makes the text stand out, prevents it from being interpreted as a label or identifier, and allows special characters like $\$, - and & to be used.

If a file is used by several modelers and is updated occasionally, a file modification log at the top of the file will be in order. It should contain the following information: the modification date, version number, modification(s) made and who made the modification. For example, a set called \texttt{version} may be used to keep track of the dates the input files were modified:

\begin{verbatim}
Set version(*,*,*,*);
   ...
version("my_file","May","19","2016") = yes;
version("my_include_file","Sep","30","2016") = yes;
   ...
display version;
\end{verbatim}

Note that the display statement will generate a display of all elements of the set \texttt{version}, each indicating on which day a component of the model was modified.

### 4.14 Fixing Compilation Errors

As detailed in chapter GAMS Output, the execution of a GAMS program passes through several stages, where the compilation is the first stage. Often when a program is run for the first time, it cannot be solved successfully because of compilation errors. This can be very frustrating, especially for new users. In this tutorial we will explain the causes of the most common compilation errors and offer some advice on how to fix them. For an introduction to compilation errors, see section Compilation Errors. In addition, the tutorial A GAMS Tutorial by Richard E. Rosenthal offers some detailed material on this topic.

Note

Frequently, many compilation errors in the latter part of the code are actually consequential errors that will disappear as soon as the compilation errors in the beginning have been resolved. Therefore we recommend to start with fixing the first few errors and to run the program again. More often than not, many subsequent compilation errors will have vanished.

#### 4.14.1 Preliminary Remarks

Before we will turn to an overview of the most common errors, examples that illustrate them and advice how to resolve them, we will provide some basic information on compilation errors and error messages.
4.14.1.1 Finding Compilation Errors

Compilation errors are marked with four asterisks (****) in the compilation output of the GAMS listing file (also called output file or lst file, since it has the extension .lst), hence it is easy to detect lines where compilation errors occurred by inspection.

Note

Compilation error messages also appear in the LOG file. And, if the GAMS IDE is used, a double-click at an error message in the LOG will navigate directly to the problematic code in the source file.

Consider the following simple example:

Set c "crops" / barley, wheat, soy, wheat, rice /

If this set statement appears in a program, the resulting compilation output will contain the following lines:

1 Set c "crops" / barley, wheat, soy, wheat, rice /
   **** $172

Error Messages

172 Element is redefined

Note that in the echo print of the input file a line starting with **** is inserted and the dollar sign $ followed by a number appears on this line. This indicates that a compilation error - in this case error 172 - was discovered in the line above. In addition, a list of all errors with explanatory error messages is given at the end of the echo print.

In our example, the error refers to wheat and the error message addresses the cause of the error: the respective element is redefined. Thus we check the other elements of the set and quickly realize that wheat appears a second time. If we delete the second instance of wheat and run the code again, this line will not cause any errors anymore. Note that many compilation errors are as easy to fix as demonstrated here.

4.14.1.2 Repositioning Compilation Error Messages

By default, the error messages are listed at the end of the echo print. GAMS allows to customize this position with the command line parameter ErrMsg: the value of 1 for ErrMsg will move the error messages directly beneath the line where the respective error is marked and the value of 2 will suppress the error messages in the lst file completely. Note that the default is 0.

Consider the following simple example.

Set a / a1 * a7 /
Set b(a) / a7 * a9 /
Parameter p(k) / i3 47 /

The resulting compilation output will contain the following lines:
4.14 Fixing Compilation Errors

1 Set a / a1 * a7 /;
2 Set b(a) / a7 * a9 /;
**** $170
3 Parameter p(k) / i3 47 /;
**** $120
...

Error Messages

120 Unknown identifier entered as set
170 Domain violation for element

Next, we run the same code with the following call:

> gams test ErrMsg=1

The resulting compilation output follows.

1 Set a / a1 * a7 /;
2 Set b(a) / a7 * a9 /;
**** $170
**** 170 Domain violation for element
3 Parameter p(k) / i3 47 /;
**** $120
**** 120 Unknown identifier entered as set

Observe that the error messages are now placed on the line after the error is marked. This can be especially useful if the program contains many lines of code.

Note

Users may change the system level defaults by entering the line ErrMsg=1 in the file.gmsprmnt.txt on Windows, or.gmsprmun.txt on Unix machines as in the following example:

******************************************************************************
* GAMS 2.50 Default Parameterfile for Windows NT                      *
* Gams Development Corp.                                              *
* Date : 20 Mar, 1998                                                 *
******************************************************************************
* entries required by CMEX, put in by gams.exe:                       *
* SYSDIR                                                             *
* SCRDIR                                                             *
* SCRIPTNEXT                                                         *
* INPUT                                                              *
errmsg=1

4.14.2 Resolving Common Compilation Errors

There are hundreds of compilation errors in GAMS, but some of them are particularly frequent. In the table below we present these common errors, with a brief description of the possible cause and a link to a subsection below where examples and more details are given. We recommend that users also read the error messages, since they often contain additional hints.
<table>
<thead>
<tr>
<th>GAMS Error</th>
<th>Common Cause of Error</th>
<th>Subsections with Examples and More Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Closing parentheses, square brackets or braces are missing.</td>
<td>H</td>
</tr>
<tr>
<td>36</td>
<td>The two dots .. are missing in the equation definition.</td>
<td>I</td>
</tr>
<tr>
<td>37</td>
<td>The equation type (e.g. =L=) is missing in the body of the equation.</td>
<td>I</td>
</tr>
<tr>
<td>51-60</td>
<td>There are prohibited nonlinear expressions.</td>
<td>J</td>
</tr>
<tr>
<td>66</td>
<td>A data item which has not been given numerical values appears in an equation.</td>
<td>K</td>
</tr>
<tr>
<td>71</td>
<td>The equation has been declared, but not defined.</td>
<td>I</td>
</tr>
<tr>
<td>96, 97</td>
<td>A statement followed by another statement is not terminated with ;.</td>
<td>B</td>
</tr>
<tr>
<td>120</td>
<td>GAMS cannot find a set with this name. Check for typos in the set name and set elements that are referenced without quotes.</td>
<td>C, L</td>
</tr>
<tr>
<td>125</td>
<td>The set is controlled more than once, e.g., by an indexed operation like sum and by an equation definition.</td>
<td>F</td>
</tr>
<tr>
<td>140</td>
<td>GAMS is looking for a keyword or declared item and cannot find it. Check spelling and declarations.</td>
<td>A, C, K, M</td>
</tr>
<tr>
<td>141</td>
<td>The parameter without data is used or problems with solve and therefore attributes .l and .m are empty.</td>
<td>K</td>
</tr>
<tr>
<td>148</td>
<td>The identifier is referenced with more or less indexed sets than in the declaration.</td>
<td>E</td>
</tr>
<tr>
<td>149</td>
<td>The set is not controlled, neither by an indexed operation like sum, nor by an equation definition, nor by a loop or similar.</td>
<td>G, L</td>
</tr>
<tr>
<td>170</td>
<td>The referenced set element cannot be found in the set defined for this index position. Check for typos, omissions in the set declaration, missing quotes and references to the wrong set.</td>
<td>C, D</td>
</tr>
<tr>
<td>171</td>
<td>A domain error. The wrong set is referenced for the respective index position.</td>
<td>E, L</td>
</tr>
<tr>
<td>195</td>
<td>The name used here was already used for another identifier.</td>
<td>N</td>
</tr>
<tr>
<td>198</td>
<td>Using the operation ord or a lag/lead operation with a set that is not ordered.</td>
<td>O</td>
</tr>
<tr>
<td>256</td>
<td>Something is wrong with the model specification. This is often a consequential error of another error. Look for other error messages immediately after the solve statement.</td>
<td>I, J</td>
</tr>
<tr>
<td>257</td>
<td>The solver is not checked. This may be a consequential error of any GAMS error.</td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>Probably the quotes in a set element reference are missing.</td>
<td>L</td>
</tr>
</tbody>
</table>
4.14 Fixing Compilation Errors

<table>
<thead>
<tr>
<th>GAMS Error</th>
<th>Common Cause of Error</th>
<th>Subsections with Examples and More Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>408</td>
<td>Surplus closing parentheses, square brackets or braces.</td>
<td>H</td>
</tr>
</tbody>
</table>

4.14.2.1 Error A: Misplaced Semicolons

Maybe the most common error for new users is related to semicolons. Consider the following example adapted from the well-known transportation model [TRNSPORT]:

Sets i "canning plants" / Seattle, San-Diego / ;
   j "markets" / New-York, Chicago, Topeka / ;

The resulting compilation output will contain the following lines:

```
20  Sets i  canning plants  / Seattle, San-Diego / ;
21   j  markets  / New-York, Chicago, Topeka / ;
****  $140  $36
...
```

Error Messages

36 '=' or '..' or ':=' or '$=' operator expected
   rest of statement ignored
140 Unknown symbol

What went wrong? GAMS statements like the set statement in our example have to terminate with a semicolon, unless the next line of code begins with a reserved word. Now, set statements may extend over several lines and define several sets. In our example, the set statement extends over two lines and two sets are declared. However, there is a semicolon at the end of the first line, therefore GAMS assumes that the set statement ends there. The symbol j at the beginning of the next line has not been declared yet and it is not a GAMS keyword, thus it is marked as Unknown symbol. Note that error 36 is a consequential error that will disappear as soon as the first error has been resolved.

How do we fix this? There are two ways: either we could drop the semicolon at the end of the first line and thus indicate that the set statement continues to the second line or we could insert the keyword Set at the start of the second line and thus introduce a new set statement. These two alternatives are illustrated below:

Sets i "canning plants" / Seattle, San-Diego / ;
   j "markets" / New-York, Chicago, Topeka / ;

or

Sets i "canning plants" / Seattle, San-Diego / ;
Set j "markets" / New-York, Chicago, Topeka / ;

Note

- In general, GAMS statements have to be terminated with a semicolon. The semicolon may be omitted if the next line starts with a GAMS keyword.
- Even if it is not required, it is good practice to always end a statement with a semicolon.
4.14.2.2 Error B: Missing Semicolons

Consider the following example adapted from the well-known transportation model [TRNSPORT]:

Equations
  cost       "define objective function"
  supply(i)  "observe supply limit at plant i"
  demand(j)  "satisfy demand at market j"

  cost ..   z =e=  sum((i,j), c(i,j)*x(i,j)) ;

The resulting compilation output will contain the following lines:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Equations</td>
</tr>
<tr>
<td>53</td>
<td>cost define objective function</td>
</tr>
<tr>
<td>54</td>
<td>supply(i) observe supply limit at plant i</td>
</tr>
<tr>
<td>55</td>
<td>demand(j) satisfy demand at market j</td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;</td>
</tr>
</tbody>
</table>

**** $97 $195,96,195,96 $195,96
...

Error Messages

96 Blank needed between identifier and text
(-or- illegal character in identifier)
(-or- check for missing ';' on previous line)

97 Explanatory text can not start with '$', '=' or '..'
(-or- check for missing ';' on previous line)

What went wrong? GAMS statements like the equation declaration in our example have to terminate with a semicolon, unless the next line of code begins with a GAMS keyword. In our example, we omitted the semicolon at the end of the equation declaration statement and started a new statement after a blank line.

How do we fix this? We simply add a semicolon after the last explanatory text at the end of the equation declaration statement.

Note

A missing semicolon is often associated with error 96 or 97.

4.14.2.3 Error C: Spelling Mistakes

This error occurs if sets, set elements, parameters, etc. are referenced with a different name than the name they were declared with. Note that differences in capitalization are not considered spelling mistakes since GAMS is case insensitive.

Consider the following example adapted from the well-known transportation model [TRNSPORT]:

---
Sets i "canning plants" / Seattle, San-Diego /  
   j "markets" / New-York, Chicago, Topeka / ;

Parameters a(i) "capacity of plant i in cases"  
   / Seatle 350  
       san-diego 600 / ;

Equations  
  cost "define objective function"  
  supply(i) "observe supply limit at plant i"  
  demand(j) "satisfy demand at market j" ;

cst .. z =e= sum((i,j), c(i,j)*x(i,j)) ;

The resulting compilation output will contain the following lines:

20 Sets i canning plants / Seattle, San-Diego /  
21   j markets / New-York, Chicago, Topeka / ;
22
23 Parameters a(i) capacity of plant i in cases  
24  / Seatle 350  
**** $170  
25 san-diego 600 / ;
...
49 Equations  
50 cost define objective function  
51 supply(i) observe supply limit at plant i  
52 demand(j) satisfy demand at market j ;
53
54 cst .. z =e= sum((i,j), c(i,j)*x(i,j)) ;
**** $140  
...
Error Messages

140 Unknown symbol
...
170 Domain violation for element

What went wrong? Note that in line 24 the set element Seattle was referenced as Seatle. GAMS does not recognize this symbol as an element of the set i and thus marks this with error 170: Domain violation. GAMS catches this error since it automatically performs domain checking. The typo in line 54, where the equation cost is referenced as cst, is marked with error 140 (Unknown symbol), since cst was not declared before.

Note

Set name misspellings are associated with error 120, set element misspellings with 170 and other misspellings with 140.

Fixing errors like these is as easy as fixing typos.
4.14.2.4 Error D: Missing Set Elements

Sometimes users forget to include an element in a set statement, but reference it later. Consider the following simple example:

```
Set       c "crops"       / barley, wheat, soy /;
Parameter d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
```

The resulting compilation output will contain the following lines:

```
1 Set       c "crops"       / barley, wheat, soy /;
2 Parameter d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
**** $170
...
Error Messages

170 Domain violation for element
```

What went wrong? Note that the symbol *rice* was not defined as an element of the set *c*, thus GAMS does not recognize it and marks it with error 170: *Domain violation*. GAMS catches this error since it automatically performs *domain checking*.

Note

If symbols are not defined as set elements, but are referenced as if they belong to the set later, error 170 will occur.

This is easy to fix: we just add the missing element(s) to the elements of the respective set.

4.14.2.5 Error E: Problems with Indices

Recall that variables, sets, parameters and equation may be defined over one or more indices. If the identifiers are referenced later in the program, the indices must appear exactly in the order that was specified in the respective definition statement. Consider the following example adapted from the well-known transportation model [TRNSPORT]:

```
Sets i "canning plants" / seattle, san-diego /
     j "markets"       / new-york, chicago, topeka / ;
Table d(i,j) "distance in thousands of miles"
        new-york  chicago  topeka
seattle   2.5      1.7      1.8
san-diego  2.5      1.8      1.4   ;
Scalar f "freight in dollars per case per thousand miles" /90/ ;
Parameter c(i,j) "transport cost in thousands of dollars per case";
c(i,j,j) = f * d(j,i) / 1000 ;
```

The resulting compilation output will contain the following lines:
4.14 Fixing Compilation Errors

Parameter \( c(i,j) \) transport cost in thousands of dollars per case;
\[ c(i,j,j) = f \times d(j,i) / 1000; \]

Error Messages

148 Dimension different - The symbol is referenced with more/less indices as declared
171 Domain violation for set

What went wrong? Note that the parameter \( c \) was declared over two indices, but in the assignment statement in line 41 it is referenced with three indices. Such a mistake is marked with error 148. Note further, that the indices of \( d \) are referenced in the wrong order, which is marked with error 171.

Note

If an identifier is referenced with more or less indices as declared, then the error 148 will be triggered.
If the indices are in the wrong order, the error 171 will be triggered.

This is easy to fix: we just check the declaration or definition statement(s) and adjust the reference.

Observe that the domain error 171 is also triggered if an identifier is referenced with index say \( i \), but was defined over say \( j \). However, there will be no domain error if \( i \) is a subset of \( j \) or \( i \) and \( j \) reference the same set since they are aliases.

4.14.2.6 Error F: Summing over Sets that are Already Indexed

Consider the following equation definition that is adapted from the well-known transportation model [TRNSPORT]:

\[
supply(i) .. \sum((i,j), x(i,j)) =l= a(i); \]

The resulting compilation output will contain the following lines:

59 \[ supply(i) .. \sum((i,j), x(i,j)) =l= a(i); \]

Error Messages

125 Set is under control already

What went wrong? Note that the equation is indexed over the set \( i \), therefore the indexed operation \( \sum \) in the body of the equation may not be controlled by the index \( i \) again.

Note

Summing over sets that are already indexed will trigger error 125.

How do we fix this? We need to carefully check the indexed operation and drop the surplus index. In other cases the controlling index may have to be dropped from the equation name. Note that an error like this is often indicative of a lack of clarity in thinking.
4.14.2.7 Error G: Uncontrolled Sets

Consider the following equation definition that is adapted from the well-known transportation model [TRNSPORT]:

\[
\text{demand .. } \sum(i, x(i,j)) \geq b(j) \ ;
\]

The resulting compilation output will contain the following lines:

```
61 demand .. sum(i, x(i,j)) =g= b(j) ;
**** $149 $149
...
```

Error Messages

149 Uncontrolled set entered as constant

What went wrong? Note that the variable \(x\) is indexed over the sets \(i\) and \(j\), but, while \(i\) is the index of the indexed operation \(\text{sum}\), \(j\) is "free": it does neither appear as index of the equation nor as controlling index in \(\text{sum}\). Therefore it was entered as if it were a constant. Note further, that \(j\) is also used as an index on the right-hand side of the equation without being controlled.

Note

Error 149 marks instances when an index is not controlled in the context of an equation or an indexed operation like \(\text{sum}\) or \(\text{prod}\).

How do we fix this? We have to think what we actually want to model. In this case we need to enter \(j\) as a controlling index for the equation. But it would also be possible to add it as a controlling index for \(\text{sum}\) (but not both!). In the latter case there would still be the error on the right-hand side which needed to be taken care of with another indexed operation, e.g., \(\text{sum}\) again. Both approaches would remove the compilation error, but one has to think about what actually should be modeled to decide which is the right fix here.

4.14.2.8 Error H: Mismatched Parentheses

Consider the following two equations adapted from the model [CHENERY]:

\[
\text{mb}(i) \cdot x(i) = g = y(i) + \sum(j, aio(i,j) \cdot x(j)) + (e(i) - m(i)) \cdot t(i) ;
\]

...  
\[
\text{dvv}(i) \cdot (\text{sig}(i) < 0) \cdot \text{v}(i) = e = (\pi \cdot (1 - \text{del}(i)) / \text{del}(i)) \cdot (-\rho(i) / (1 + \rho(i))) ;
\]

The resulting compilation output will contain the following lines:

```
137 mb(i) .. x(i) =e= y(i) + sum(j, aio(i,j)*x(j)) + ( e(i) - m(i) )*$t(i) ;
**** $408,409
...
155 dvv(i)$(sig(i) <> 0).. vv(i) =e= (pi*(1-del(i))/del(i))**(-rho(i)/(1+rho(i))) ;
**** $8
...
```

Error Messages

8 ')' expected
...
408 Too many ),] or }
409 Unrecognizable item - skip to find a new statement looking for a ';' or a key word to get started again
What went wrong? Note that in the first equation we have one surplus closing parenthesis, this is marked with error 408. The error marked with 409 is a consequential error, it will disappear once the first error has been fixed. In the second equation, one closing parenthesis is missing resulting in error 8.

Attention

Opening and closing parentheses ( ), square brackets [ ] and braces (curly brackets) { } must match.

Note

Surplus closing parentheses, bracketes and braces are marked with error 408 and missing closing parenthesis, brackets and braces are marked with error 8. Missing opening parentheses, brackets and braces may result in GAMS marking surplus closing parentheses, brackets and braces; surplus opening parentheses, brackets and braces may result in GAMS marking missing closing parentheses, brackets and braces.

While fixing errors like these entails carefully counting opening and closing parentheses, there are strategies that help to prevent mismatching parentheses errors. Many text editors (like the GAMS IDE) offer a feature that identifies matching parentheses and will issue a warning if there is a mismatch. We recommend to use this feature. Further, we recommend to also use the alternatives to parentheses: square brackets and braces. They are especially useful if there are several opening parentheses since it is easier to determine by inspection if each has a matching closing symbol.

4.14.2.9 Error I: Mistakes Relating to Equations

Recall that each declared equation must be defined if it is to be used in a model statement. If the equation definition is missing, GAMS will mark an error beneath the solve statement that refers to a model which references the respective undefined equation. Consider the following example adapted from the well-known transportation model [TRNSPORT]:

Equations

\begin{align*}
\text{cost} & \quad \text{"define objective function"} \\
\text{supply}(i) & \quad \text{"observe supply limit at plant i"} \\
\text{demand}(j) & \quad \text{"satisfy demand at market j"} \\
\end{align*}

\begin{align*}
\text{cost} .. & \quad z =e= \sum((i,j), c(i,j)*x(i,j)) \\
\text{supply}(i) .. & \quad \sum(j, x(i,j)) =l= a(i) \\
\end{align*}

Model transport /all/ \\
solve transport using lp minimizing z \\

The resulting compilation output will contain the following lines:

60 Model transport /all/ \\
61 solve transport using lp minimizing z \\
**** $71,256 \\
**** The following LP errors were detected in model transport: \\
**** 71 demand is an undefined equation \\
... \\
Error Messages

71 The symbol shown has been declared as an equation, but no
  Symbolic equation (..) was found. hint - look for commas in the
  Documentation text for the equations. use quotes around the
  Text or eliminate the commas.
256 Error(s) in analyzing solve statement. More detail appears
  Below the solve statement above
What went wrong? Note that the equation demand was declared and included in the model transport, but it was not defined. GAMS detects that the equation definition is missing when it compiles the solve statement that relates to the model transport. Therefore the error 71 appears only there. The error 256 is a consequential error, it will disappear once the missing equation definition has been added (before the solve statement).

Note

Error 71 marks a missing equation definition. The message associated with this error is a nice example of an error message that is very descriptive and offers advice on how to fix the error.

Apart from forgetting to define a declared equation and forgetting to terminate the equation definition statement with a semicolon (see above), two other mistakes relating to equations are frequent: omitting the syntax element .. after the name of the equation and not properly defining the equation type in the equation definition statement. The following simple example adapted from the portfolio analysis model [ALAN] illustrates these mistakes:

\[
\begin{align*}
\text{fsum} \sum(i, x(i)) & =e= 1.0; \\
\text{dmean..} \sum(i, \text{mean}(i)*x(i)) & = \text{target};
\end{align*}
\]

The resulting compilation output will contain the following lines:

\[
\begin{align*}
45 & \text{fsum } \sum(i, x(i)) & =e= 1.0; \\
**** & $36 \\
46 & \text{dmean.. } \sum(i, \text{mean}(i)*x(i)) & = \text{target}; \\
**** & $37 \\
...
\end{align*}
\]

```
 Error Messages
36 '=' or '..' or ':=' or '$=' operator expected
   rest of statement ignored
37 '='=l=' or '='e=' or '='g=' operator expected
```

What went wrong? Note that in the first equation the two dots .. are missing, which is marked with error 36, and in the second equation the equation type is not properly specified, which leads to error 37.

4.14.2.10 Error J: Issues with Nonlinear Expressions

The compilation error messages 51 to 60 refer to issues with nonlinear expressions involving variables in equations. For example, nonlinear expressions in an LP model will cause errors of this sort. So will discontinuous functions with endogenous arguments in an NLP model. For information on endogenous arguments in functions, see section Functions in Equation Definitions. For an overview of GAMS model types, see section Classification of Models.

Consider the following simple artificial example:

```
Variables x, y, z;
Equations eq1, eq2;

eq1.. x**2 - y =e= z ;
eq2.. min(x,y) =l= 20 ;

Model silly / all /;
solve silly using lp maximizing z ;
```
The resulting compilation output will contain the following lines:

```
7  Model silly / all /;
8  solve silly using lp maximizing z;
**** $54,51,256
**** The following LP errors were detected in model silly:
**** 54 equation eq1.. VAR operands for **
**** 51 equation eq2.. the function MIN is called with non-constant arguments
```

**Error Messages**

51 Endogenous function argument(s) not allowed in linear models
54 Endogenous operands for ** not allowed in linear models
256 Error(s) in analyzing solve statement. More detail appears
Below the solve statement above

Note that error 54 marks the nonlinear expression $x^2$ in a linear model and error 51 refers to the discontinuous function $\min$ that may be used with variables only in models of the type DNLP. The error 256 is a consequence of the two previous errors.

### 4.14.2.11 Error K: Using Undefined Data

Referencing data items that were not declared with a scalar, parameter or table statement will cause error 140: GAMS will indicate that the respective symbol is unknown. More often data items are declared, but not defined, i.e. they are not initialized with values.

Consider the following example adapted from the well-known transportation model [TRNSPORT]. Assume that the scalar $f$ is declared, but has no numerical value. The resulting compilation output will contain the following lines:

```
40  Scalar f freight in dollars per case per thousand miles;
41
42  Parameter c(i,j) transport cost in thousands of dollars per case;
43
44   c(i,j) = f * d(i,j) / 1000;
**** $141
...
61  Model transport /all/;
62  solve transport using lp minimizing z;
**** $257
63
64  display x.l, x.m;
**** $141
...
```

**Error Messages**

141 Symbol declared but no values have been assigned. Check for missing data definition, assignment, data loading or implicit assignment via a solve statement.
A wild shot: You may have spurious commas in the explanatory text of a declaration. Check symbol reference list.
257 Solve statement not checked because of previous errors
What went wrong? Note that the scalar $f$ was declared, but there was no value assigned to it. Thus, when it is referenced in the calculation in line 44, the "empty" scalar causes error 141. Error 257 is a consequential error and leads to the second error 141. This error occurs in the context of variable attributes in the display statement. Recall that variable attributes are data associated with variables. As the solve statement could not be checked, the level and marginal values of the variable $x$ are not defined and hence error 141 follows.

Note

Error 141 marks places where data items, that have not been defined, are referenced in computations in the context of assignment statements. It also marks places where undefined variable (or equation) attributes are referenced.

In addition, undefined data items may be referenced in the context of equation definitions. Suppose we declare the parameter $c$ in model [TRANSPORT], but forget the corresponding assignment statement. Then the compilation output will contain the following lines:

57    cost ..      z =e=  sum((i,j), c(i,j)* x(i,j)) ;
...  
62    solve transport using lp minimizing z ;
**** $66,256
**** The following LP errors were detected in model transport:
**** 66 equation cost.. symbol "c" has no values assigned
...

Error Messages

66 The symbol shown has not been defined or assigned
A wild shot: You may have spurious commas in the explanatory text of a declaration. Check symbol reference list.
256 Error(s) in analyzing solve statement. More detail appears
Below the solve statement above

Note that the missing data in the body of the equation cost is marked with error 66 and the following error is a consequential error which will disappear once 66 has been fixed.

Note

Referencing undefined data items in the body of an equation causes error 66.

4.14.2.12 Error L: Referencing Set Elements Without Quotes

Recall that if set elements are referenced, they need to appear in single or double quotes. Omitting the quotes may cause different errors depending on the context. The following example is adapted from the well-known transportation model [TRANSPORT]. Suppose we start by defining the two sets and the two parameters and then add the following statements:

Scalar s;
s = a(seattle);

The resulting compilation output will contain the following lines:
4.14 Fixing Compilation Errors

37  Scalar s;
38  s = a(seattle);
**** $120,340,149,171
...
Error Messages

120  Unknown identifier entered as set
149  Uncontrolled set entered as constant
171  Domain violation for set
340  A label/element with the same name exist. You may have forgotten
to quote a label/element reference. For example,
set i / a,b,c /; parameter x(i); x('a') = 10;

What went wrong? On the right-hand side of the assignment statement we have a reference with missing quotes. Note that this creates four (!) errors:

- 120 - GAMS expects a controlling set and Seattle is not recognized and therefore considered an unknown identifier.
- 340 - Then GAMS realizes that actually there is a label called Seattle in the program and helpfully offers a hint: You may have forgotten to quote a label/element reference.
- 149 - See section Error G: Uncontrolled Sets above.
- 171 - See section Error E: Problems with Indices above.

This is a nice example for one mistake triggering several errors that may look daunting. However, fixing them is as easy as adding single or double quotes.

4.14.2.13 Error M: Missing Declaration Statements

Recall that sets, parameters, variables and equations have to be declared before they may be used in definition or assignment statements. Suppose the equation demand in the well-known transportation model [TRNSPORT] was defined but not declared. Then the compilation output would contain the following lines:

59  demand(j) .. sum(i, x(i,j)) =g= b(j) ;
**** $140
...
Error Messages

140  Unknown symbol

Note that as demand was not declared, GAMS does not recognize it and therefore marks it with error 140: Unknown symbol.

Note

If error 140 is reported, the respective declaration statement is probably missing. However, the underlying cause may also be a simple typo.
4.14.2.14 Error N: Using the Same Name for Different Identifiers

Recall that each GAMS identifier must have a unique name. If the same name is used more than once, an error will be triggered. Consider the following modification of the transportation model [TRNSPORT]:

Parameters  capacity(i)  "capacity of plant i in cases"
            / Seattle 350, San-Diego 600 /
            demand(j)  "demand at market j in cases"
            / New-York 325, Chicago 300, Topeka 275 /;

Equations  cost "define objective function"
           supply(i) "observe supply limit at plant i"
           demand(j) "satisfy demand at market j" ;

The resulting compilation output will contain the following lines:

47 Equations
48   cost define objective function
49   supply(i) observe supply limit at plant i
50   demand(j) satisfy demand at market j ;

**** $195
...

Error Messages

195 Symbol redefined with a different type

What went wrong? Note that demand was first declared as a parameter and later as an equation. GAMS will mark this repeated use of the same name with error 195.

4.14.2.15 Error O: Using ORD with an Unordered Set

Recall that the operation ord is only defined for ordered sets. If ord is used with an unordered set, error 198 will be triggered.

Consider the following example:

Set   a  "a couple of the elements" / r2, r3 /;
Set   b  "more elements" / r1*r4 /
Scalar c  "counter" / 0 /;

loop( b $ (ord(b) > 3),
     c = c + 1 );
display c;

The resulting compilation output will contain the following lines:
Error Messages

198 Set used in 'ord' or lag is not ordered.
    Hint: Some of the elements of the set were used before this
    was initialized and the order was different from the order used
    in this set. Try to initialize the set earlier.
    $offOrder allows lag operations on dynamic sets, reset with
    $onOrder

What went wrong? Two elements of the set \( b \) featured already in set \( a \) that was defined before set \( b \). Therefore the order of the elements in set \( b \) is as follows:

\[ r2 \quad r3 \quad r1 \quad r4 \]

Obviously, this set is not ordered and hence GAMS marks error 198 when the operator \( \text{ord} \) is applied to it.

How do we fix this? In this case it is easy: we just define the set \( b \) before set \( a \) and \( b \) will be an ordered set.

Consider one last example:

```
Set a "all elements" / r1*r10 /;
Set b "elements in different order" / r3, r1, r4, r7 /;
Scalar c "counter" / 0 /;
```

```
loop( b $ (ord(b) > 3),
    c = c + 1 );
display c;
```

The resulting compilation output will contain the following lines:

```
5 loop( b $ (ord(b) > 3),
   **** $198
6 c = c + 1 );
```

What went wrong? Note that even though all elements in set \( b \) are also elements of set \( a \), they are not specified in the same order. Therefore the set \( b \) is unordered. For more details, see section Ordered and Unordered Sets.

4.15 Finding and Fixing Execution Errors and Performance Problems

This tutorial is structured in four main parts: we will discuss how to find and resolve errors that are detected during execution of a GAMS model, we will give some guidance for model development and debugging and we will present techniques to increase efficiency by reducing GAMS execution time and memory use.
4.15.1 Resolving Execution Errors

Recall that GAMS passes through a program file several times in the process of generating and solving a model. Errors may occur in each phase. In this section we will give some guidance on how to resolve errors that occur during execution, so after compilation. For advice on resolving compilation errors, see the tutorial Fixing Compilation Errors. For more information on the process of generating and solving a model in GAMS, see the introduction to chapter GAMS Output.

At execution, several things could go wrong and cause an error. We will look at these potential error sources separately in this section. First we look at arithmetic errors and exceeded internal limits during data manipulation, we will continue with problems during model generation and model solution. At the end, we will briefly discuss how execution errors may be managed with the function execError.

4.15.1.1 Arithmetic Errors

GAMS execution errors may be caused by illegal arithmetic operations like a negative argument for log, division by zero and exponentiation where the base is a negative number. The following simple example serves as illustration:

```
Set   s   / s1*s5 / ;
Parameter p(s) "data to be exponentiated"
           d(s) "divisors"
           r(s) "result";
p   = 1;
p("s2") = -1;
d   = 1;
d("s3") = 0;
r   = p(s)**2.1 / d(s)
display r;
```

The first sign that something in the execution went wrong is the following flag in the log output:

```
*** Status: Execution error(s)
```

The resulting execution output will contain the following lines:

```
Execution
**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)
---- 11 PARAMETER r result
s1 1.000, s2 UNDF, s3 UNDF, s4 1.000, s5 1.000
```

Observe that the execution output begins with two error messages that can be easily found since they are marked with four asterisks ****. The error messages are very informative: they indicate the line where the errors occurred and provide details about the nature of the errors. Further, the output generated by the display statement shows that the errors occurred when the values for r("s2") and r("s3") were computed. Inspecting the assignment statement for these two values, we realize that in the first instance
the base for the exponentiation is -1, which obviously is a negative number and hence is not allowed in this operation. In the second instance, the problem is that we divide by $d("s3")$ which equals zero.

In this example, the errors are easily resolved with data revisions. In general, we recommend to use conditional assignments to prevent errors like these.

Note that in the example above the error messages indicated exactly where the problem was and it was easy to find the cause of the error. However, this is not always the case. In particular, if the problem is within a multi-dimensional item the user will need more patience. Usually it helps to display the results of the problematic operation and look for faulty entries. In addition, displaying the input data to the respective operation will help to investigate the numerical properties of the data that was entered in the computation. Often more displays will be needed to trace faulty input data through the program. Eventually this will lead the user to understand why the data has taken on the specific numerical values it has.

4.15.1.2 Exceeding GAMS Limits

By default, GAMS stops the solve of a model after 1e10 seconds (wall clock time) or 2e9 iterations. These limits may be adjusted with the options reslim and iterlim respectively. Note that both options are also available as command line parameters and model attributes. In addition, the workspace may be limited with the command line parameters WorkFactor and Workspace. Note that these options are also available as model attributes. If any of these limits are exceeded, the execution of the solve statement will be interrupted.

For example, we could add the following option statement somewhere before the solve statement in the production model [CHENERY]:

```plaintext
option iterlim = 20;
```

Note that this statement reduces the iteration limit to just 20. The log output will contain the following lines:

```
** Feasible solution. Value of objective = 1033.34069261
** The iteration limit has been reached.
--- Restarting execution
--- chenery.gms(228) 2 Mb
--- Reading solution for model chenrad
*** Status: Normal completion
--- Job chenery.gms Stop 11/21/16 16:52:43 elapsed 0:00:00.106
```

Also the solve summary in the listing file notes the interrupt:

```
SOLVE SUMMARY
MODEL       chenrad       OBJECTIVE       td
TYPE        NLP            DIRECTION       MAXIMIZE
SOLVER      CONOPT         FROM LINE       228

**** SOLVER STATUS  2 Iteration Interrupt
**** MODEL STATUS   7 Feasible Solution
**** OBJECTIVE VALUE 1058.9199
RESOURCE USAGE, LIMIT 0.078 1000.000
ITERATION COUNT, LIMIT 20 20
EVALUATION ERRORS 0 0
```
Observe that the solver status **Iteration Interrupt** indicates that the execution terminated because the iteration limit has been reached resulting in a feasible solution, but not the optimal solution. The line **ITERATION COUNT, LIMIT ...** reports that 20 iterations were performed and that 20 was also the limit for the number of iterations. Setting `iterlim` to a larger value will resolve this issue.

Similarly, allocating too little work space will cause the solver to terminate with no solution. For example, restricting the work space for the nonlinear test model [MHW4D] to just 0.1 MB and running it with the solver MINOS will produce the following lines in the log output:

```
Work space requested by solver -- 0.77 Mb
Work space requested by user -- 0.10 Mb
Work space allocated -- 0.10 Mb

EXIT - Not enough storage to solve the model.
    Request at least 0.19 Mbytes.
```

The **solve summary** in the listing file will contain the following information:

```
SOLVE SUMMARY

MODEL wright
TYPE NLP
SOLVER MINOS

**** SOLVER STATUS 9 Setup Failure
**** MODEL STATUS 13 Error No Solution
```

Note that increasing the work space to at least the minimum amount requested by the solver will resolve this issue.

When dealing with large nonlinear expressions defined over a very large domain, one can face the following error.

```
*** Status: Terminated due to limits in NLCodeAdd
*** Cannot handle more than 2147483647 instruction in NL code
*** Inspect listing file for more information
```

The first thing to check in this case is correctness of the model i.e., if you are not generating anything more than necessary. If this is not the case and the model is indeed large, a generic advise is to introduce intermediate variables to get a smaller code size per block of equations. One can also consider partitioning the domains so that one can have more number of smaller blocks.
4.15.1.3 Resolving Model Generation Errors

Further execution errors may be detected when GAMS is generating the model before passing it to the solver. These errors may be arithmetic errors in the body of equations or errors in the structure of the model that cause the model to be inherently infeasible.

Consider the following simple example with arithmetic errors in the body of the equations. They are similar to the errors in the assignment in the example in section Arithmetic Errors above.

```
Set s / s1*s5 / ;
Parameter p(s) "data to be exponentiated"
    d(s) "divisors"
    m(s) "multipliers";
p(s) = 1;
p("s2") = -1;
d(s) = 1;
d("s3") = 0;
m(s) = 1;
m("s4") = 0;
Positive variable x(s);
Variable z;
Equations obj "objective function"
    xlim;
    obj.. z =e= sum(s,p(s)**2.2*x(s));
    xlim(s).. m(s) / d(s)*x(s) =e= 1;
Model mymodel / all /;
solve mymodel using lp maximizing z;
```

If we run this model, the log output will contain the following lines:

```
*** SOLVE aborted
--- Executing CPLEX: elapsed 0:00:00.006
--- test.gms(23) 4 Mb 3 Errors
*** Status: Execution error(s)
--- Job test.gms Stop 11/21/16 19:10:12 elapsed 0:00:00.006
```

Observe that the solve was aborted since there are 3 execution errors. The equation listing in the listing file will contain further details about these execution errors:

```
Equation Listing  SOLVE mymodel Using LP From line 23

**** Exec Error at line 19: rPower: FUNC DOMAIN: x**y, x < 0
---- obj =E= objective function
obj.. - x(s1) + UNDF*x(s2) - x(s3) - x(s4) - x(s5) + z =E= UNDF ; (LHS = UNDF)

**** Exec Error at line 20: division by zero (0)
**** Exec Error at line 20: Equation infeasible due to rhs value
```
Note that there is an arithmetic error relating to exponentiation in the first equation and an arithmetic error and an infeasibility in the second equation.

In our example, it was easy to detect the execution errors and their cause. However, an error in a multi-dimensional equation block may be much more difficult to find. Note that by default, only the first three entries in each equation block are shown in the equation listing. We recommend to use the option limrow to get a full listing, as this is the easiest way to inspect execution errors in the body of equations.

4.15.1.4 Resolving Solve Errors

In the solution phase, an external solver program processes the model and creates output with details about the solution process. Solve errors may be either function evaluation errors or presolve errors.

4.15.1.4.1 Resolving Function Evaluation Errors

Some solve statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the solution report.

Function evaluation errors are numerical errors like those discussed in section Arithmetic Errors above. Other examples include square roots of negative variables and squaring a negative term, say $x$, using the syntax $x^{2}$.

Attention

Squaring a negative term, say $x$, using the syntax $x^{2}$ will cause an error. However, the alternatives $\text{sqr}(x)$ and $x\times x$ will work (see here for an explanation).

Note that by default the solver subsystems will interrupt the solution process if arithmetic errors are encountered. Users may wish to permit a certain number of arithmetic errors and have reported error warnings instead. The option domlim facilitates this modification. Note that the default value for domlim is zero.

The best way to avoid evaluating functions outside their domain of definition is to specify reasonable variable bounds. However, there are cases when bounds are not enough. Consider the following simple example:

```
Set i / i1*i5 /;
Variables x(i), z;
Equations r1, r2(i);

r1.. z =e= log(sum(i, x(i)));
r2(i).. x(i) =l= 10;
x.lo(i) = 0;
x.l(i) = 5;

Model takelog / all /;
solve takelog using nlp minimizing z;
```
If we try to solve this little program with the solver MINOS, the log output will contain the following line:

EXIT - Function evaluation error limit exceeded.

The solution report in the listing file will have more detailed information:

```
SOLVE SUMMARY
MODEL takelog OBJECTIVE z
TYPE NLP DIRECTION MINIMIZE
SOLVER MINOS FROM LINE 12

**** SOLVER STATUS 5 Evaluation Interrupt
**** MODEL STATUS 7 Feasible Solution
**** OBJECTIVE VALUE 0.0000

RESOURCE USAGE, LIMIT 0.183 1000.000
ITERATION COUNT, LIMIT 0 2000000000
EVALUATION ERRORS 2 0

... EXIT - Function evaluation error limit exceeded.

**** ERRORS/WARNINGS IN EQUATION r1
 2 error(s): log: FUNC DOMAIN: x < 0 (RETURNED 0)

... **** REPORT SUMMARY : 1 NONOPT ( NOPT)
0 INFEASIBLE
0 UNBOUNDED
1 ERRORS ( ****)
```

Note that the solver status has a value of 5 (Evaluation Interrupt), which means that the solver has been interrupted as more evaluation errors have been encountered than specified with the option domlim. In our case domlim equals its default value zero, thus one error is enough to cause the interruption. The equation in which the evaluation error occurred and the type of error is reported a few lines later. In our example, the equation r1 is problematic, since we take the logarithm of the expression \( \sum(i, x(i)) \), an expression which may become zero.

Note that in models such as this each individual variable \( x(i) \) should be allowed to become zero, but the sum should not. This may be achieved by introducing an intermediate variable, say \( xsum \), adding a lower bound greater than zero for it and using this variable as the argument for the function \( \log \):

```
Variable xsum;
xsum.lo = 0.0001;

Equations defxsum, r1;
defxsum ..     xsum =e=  \sum(i, x(i));
r1..          z =e= \log(xsum);
```
For more information on intermediate variables, see section *Avoiding Expressions in Nonlinear Functions* in the tutorial *Good NLP Formulations*.

Observe that solvers report the type of arithmetic problem encountered and the problematic equation, but do not identify the particular offending variable or the labels in the index of an equation that cause the error. If the cause is not obvious, users will have to investigate the numerical properties of the variables, labels and parameters in the body of the respective equation. This may involve the following:

- Displaying the input data items to the nonlinear terms in the respective equation.
- Searching the solution for equations that are infeasible (INFES) and variables that are nonoptimal (NOPT) in order to see where problems are present and which variables were being manipulated at the end of the run.
- Investigating variables and equations whose level values are zero, negative or very large at the end of the run.
- Deactivating part of the code to narrow down the problem as discussed in section *Isolating Terms in Slow Statements* below.

Resolving function evaluation errors will usually entail the following techniques:

- Adding lower bounds to variables to keep them above zero.
- Adding upper bounds to variables to prevent them from getting too large.
- Reformulating the model, for example, introducing intermediate variables.
- Providing better starting points that direct the solver search to a more relevant region. See section *Specifying Initial Values* in tutorial *Good NLP Formulations* for details.
- Fixing faulty input data.

### 4.15.1.4.2 Presolve Errors

Some solvers use a pre-processing step where the program is presolved to make the main solution process faster. During this step model errors could already be discovered, as in the following example:

```plaintext
Variables    z;
Integer Variables y1,y2;
Equations    r1,r2,r3,r4;
    r1..  z=e=y1+y2;
    r2..  y1=g=0.10;
    r3..  y2=g=0.10;
    r4..  y1+y2=l=1;
Model badpresol /all/;
solve badpresol using mip maximizing z;
```

For this problem, Cplex detects in the presolve already, that there is no feasible integer solution. This is reported in the log:
Row 'r4' infeasible, all entries at implied bounds.
Presolve time = 0.00 sec. (0.00 ticks)

... 

CPLEX Error 1217: No solution exists.
Problem is integer infeasible.

Here, Cplex makes it clear, where we have a problem: Row r4 is infeasible, because all entries are at their
"implied bounds". Lets look at r2 and r3 to see what this means: These equations set a lower bound
of 0.1 for y1 and y2. Since both variables are defined as Integer Variables, they get an implicit lower
bound of 1. Given that, equation r4 must be infeasible.

4.15.1.4.3 Solver Specific Limits  Many solvers have internal limits that may be exceeded and may
cause the listing file to report an execution error. These errors may be resolved by using either GAMS
options or solver specific options to increase the respective limits. Usually, the listing file will contain
information about which options to use. Note that the solver manuals distributed with GAMS list the
options that may be specified for each solver. For example, to relax the MINOS major iteration limit, the
user may create a file named minos.opt with the following line:

Major iterations 1000

More about solver option files can be found in section The Solver Options File.

4.15.1.5 Managing Execution Errors with the Function execError

The function execError facilitates implementing procedures that manage execution errors. Consider the
following example, which is an extension of the example in section Arithmetic Errors above.

Set s / s1*s5 / ;
Parameter p(s) "data to be exponentiated"
   d(s) "divisors"
   r(s) "result";

   p(s) = 1;
p("s2") = -1;
d(s) = 1;
d("s3") = 0;
r(s) = p(s)**2.1 / d(s)
display r;

*cause z to be undefined
Scalar z;
z = 1/0;

if(execError > 0,
   r(s)$(r(s) = z) = 0;);
display r;

Observe that we introduced a new scalar z that is deliberately undefined. In the if statement that follows, we
use the function execError in the logical condition and the undefined scalar in the conditional assignment.
The if statement has the effect that undefined entries are removed from the array of the parameter r, as
illustrated in the following lines of the execution output:
Execution

**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)

----   11 PARAMETER r result
s1  1.000,  s2 UNDF,  s3 UNDF,  s4  1.000,  s5  1.000

**** Exec Error at line 16: division by zero (0)

----  20 PARAMETER r result
s1  1.000,  s4  1.000,  s5  1.000

In addition, the function execError may be used to reset the count of the number of execution errors. Typically, it is reset to zero so that GAMS will terminate with the status message Normal completion. For example, we could add the following line at the end of the code in the example above:

execError = 0;

Note

Setting execError = 0; will not only result in a normal completion in the example above. A solve statement will not be executed if there were execution errors before by default. Setting execError = 0; before the solve statement, will allow to execute it again.

Setting execError = 0; results also in a notification in the log:

*****************************************************
*** Errors have been cleared ***
*****************************************************
*** Status: Normal completion

4.15.2 Small to Large: Aid in Development and Debugging

Many GAMS users are overly impressed with how easily GAMS handles large models. Modelers often feel such a facility means they should always work on the full model. The result is often a large, sometimes extremely large, model in the early stages of model development. Debugging such large formulations is not easy.

The algebraic modeling style employed in GAMS is inherently expandable. This offers interesting possibilities in terms of the strategy that may be employed for model development and debugging which are discussed herein.
4.15 Finding and Fixing Execution Errors and Performance Problems

4.15.2.1 An Illustrative Example

The set based algebraic modeling style implemented in GAMS is by its very nature easy to expand. It is easy to use the same model formulation on differently sized data sets. We will illustrate this based on the transportation model [TRNSPORT]. Note that we included some post-solution calculations at the end.

* Data section
Sets i "canning plants" / Seattle, San-Diego /
   j "markets" / New-York, Chicago, Topeka / ;

Parameters a(i) "capacity of plant i in cases"
   / Seattle 350, San-Diego 600 /
   b(j) "demand at market j in cases"
   / New-York 325, Chicago 300, Topeka 275 / ;

Table d(i,j) "distance in thousands of miles"
    New-York   Chicago  Topeka
Seattle     2.5      1.7     1.8
San-Diego   2.5      1.8     1.4

Scalar f "freight in dollars per case per thousand miles" / 90 / ;

Parameter c(i,j) "transport cost in thousands of dollars per case"
   c(i,j) = f * d(i,j) / 1000 ;

* Model Section
Positive Variable x(i,j) "shipment quantities in cases"
Variable z "total transportation costs in thousands of dollars"

Equations cost "define objective function"
   supply(i) "observe supply limit at plant i"
   demand(j) "satisfy demand at market j"

   cost .. z =e= sum((i,j), c(i,j) * x(i,j)) ;
   supply(i) .. sum(j, x(i,j)) =l= a(i) ;
   demand(j) .. sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Parameter m(*,*) "commodity movement";
   m(i,j) = x.l(i,j);
   m("total",j) = sum(i, x.l(i,j));
   m(i,"total") = sum(j, x.l(i,j));
   m("total","total") = sum(j, m("total",j));
option decimals = 0;
display m;

This model may be easily extended by adding more data:

* Data section
Sets i "canning plants" / Seattle, San-Diego, Baltimore, Dallas /
   j "markets" / New-York, Chicago, Topeka, Boston, Miami /;
Parameters  

a(i) "capacity of plant i in cases"
/ Seattle 350, San-Diego 600, Baltimore 450, Dallas 750 /

b(j) "demand at market j in cases"
/ New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table  
d(i,j) "distance in thousands of miles"

<table>
<thead>
<tr>
<th></th>
<th>New-York</th>
<th>Chicago</th>
<th>Topeka</th>
<th>Boston</th>
<th>Miami</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
<td>3.1</td>
<td>3.3</td>
</tr>
<tr>
<td>San-Diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
<td>3.0</td>
<td>2.7</td>
</tr>
<tr>
<td>Baltimore</td>
<td>0.2</td>
<td>0.7</td>
<td>1.8</td>
<td>0.4</td>
<td>1.1</td>
</tr>
<tr>
<td>Dallas</td>
<td>1.5</td>
<td>0.9</td>
<td>0.5</td>
<td>1.8</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Scalar f "freight in dollars per case per thousand miles" /90/;

Parameter  
c(i,j) "transport cost in thousands of dollars per case";
c(i,j) = f * d(i,j) / 1000;

* Model Section

Positive Variable  
x(i,j) "shipment quantities in cases";

Variable  
z "total transportation costs in thousands of dollars";

Equations  
cost "define objective function"
supply(i) "observe supply limit at plant i"
demand(j) "satisfy demand at market j";

cost ..  
z =e=  
  sum((i,j), c(i,j) * x(i,j)) ;
supply(i) ..  
  sum(j, x(i,j)) =l=  
  a(i) ;
demand(j) ..  
  sum(i, x(i,j)) =g=  
  b(j) ;

Model transport /all/;
solve transport using lp minimizing z;

Parameter  
m(*,*) "commodity movement";
m(i,j) = x.l(i,j);
m("total",j) = sum(i, x.l(i,j));
m(i,"total") = sum(j, x.l(i,j));
m("total","total") = sum(j, m("total",j));
option decimals = 0;
display m;

Observe that the two sets (i and j) were enlarged, the capacity (a) and demand (b) data were expanded to cover the new plant and market entries and the distance table (d) was adjusted accordingly. However, the data calculation, equations, model definition, model solution and report writing sections are identical in the two models.

4.15.2.2 Motivation and Step by Step Guide

As we have demonstrated in the example above, GAMS allows the model structure, calculations and report writing to be developed and implemented using a small data set, that may be easily expanded to larger data sets. Thus, we strongly recommend to start with a representative purposefully small data set and enlarge it to its full size once the work of model development, testing and debugging has been done. In short: work from small to large.

The larger the model the longer it takes to compile it, generate the model, execute and solve it. Generally, time expands exponentially. Working with a large model from the start will often lead to frustration even when the user is trying to find some relatively small data problems.
If a model that has already been completed needs some modification, it will be tempting to use the large data set instead of developing the modifications on a small data set. We strongly advise to use a small data set in this case, as experience shows that this way a considerable amount of time may be saved.

We recommend to follow these steps in model development:

1. Set up a small data set representing the full model with all structural features, set names, parameters etc.
2. Implement all data calculations, model features and report writing calculations.
3. Test the results of step 2 thoroughly.
4. Save the small model. Then implement a larger version with the full data set. Create separate files for data, calculation, model definition and report writing to maintain size independence. Use include files or the save and restart feature.
5. Test the larger model. Use the modeling techniques discussed below to facilitate your work.
6. Keep the small model current. As additional structural features are added to the large model, use it to test them. See section Introducing Strategical Subsets below for an easy way to maintain a small model.

4.15.2.3 Modeling Techniques

If users follow the steps for model development outlined in section Motivation and Step by Step Guide above, they will notice that it will not always be possible to model every needed feature with the small model. It is important to carefully choose the small data set so that it has all features of the larger data set. However, occasionally the peculiarities and interrelationships of the full data set cannot be reproduced in the small data set. In this section we will introduce some modeling techniques for finding problems that arise only when the full data set is used. They include saving and restarting to isolate the problem area, strategically introducing subsets and data reduction.

4.15.2.3.1 Isolating Problem Areas through Saving and Restarting

Suppose we have a model with a large data set that takes several hours to run and we wish to add some lines of code in a relatively small segment. The best way to do this is by isolating the relevant part. Isolating the part we wish to modify makes it possible to do tests and repairs without having to input data, do initial calculations and solve the whole model with each run. We recommend to use save and restart files.

For example, in chapter The Save and Restart Feature we demonstrate how to split the transportation model [TRNSPORT] in three parts: the file tranmodel.gms contains the data and the model, the file transolve.gms contains the solve statement and the file tranreport.gms contains a display statement. To run the whole model we use the following sequence, saving and restarting from the saved file:

```gams
> gams tranmodel s=s1
> gams transolve r=s1  s=s2
> gams tranreport r=s2
```

Assume we want a more elaborate report than just the display of some level values. As the file tranreport.gms contains the code relevant for reporting, we will modify only this file. Then we will test the result by running only this file, restarting from s2, without having to solve the whole model repeatedly.
4.15.2.3.2 Introducing Strategical Subsets  When full data sets are used in debugging or development, it is often helpful to narrow the focus on a few items in a set by introducing subsets. The following example is a modified version of the extended transportation model from section An Illustrative Example above.

* Data section

Sets i "canning plants" / Seattle, San-Diego, Baltimore, Dallas /
  j "markets" / New-York, Chicago, Topeka, Boston, Miami / ;

Sets plants(i) "a reduced set of canning plants"
  / Seattle, San-Diego /
  markets(j) "a reduced set of demand markets"
  / New-York, Chicago, Topeka / ;

*plants(i) = yes; markets(j) = yes;

Parameters a(i) "capacity of plant i in cases"
  / Seattle 350, San-Diego 600, Baltimore 450, Dallas 750 /
  b(j) "demand at market j in cases"
  / New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table d(i,j) "distance in thousands of miles"

<table>
<thead>
<tr>
<th></th>
<th>New-York</th>
<th>Chicago</th>
<th>Topeka</th>
<th>Boston</th>
<th>Miami</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
<td>3.1</td>
<td>3.3</td>
</tr>
<tr>
<td>San-Diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
<td>3.0</td>
<td>2.7</td>
</tr>
<tr>
<td>Baltimore</td>
<td>0.2</td>
<td>0.7</td>
<td>1.8</td>
<td>0.4</td>
<td>1.1</td>
</tr>
<tr>
<td>Dallas</td>
<td>1.5</td>
<td>0.9</td>
<td>0.5</td>
<td>1.8</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Scalar f "freight in dollars per case per thousand miles" /90/ ;

Parameter c(i,j) "transport cost in thousands of dollars per case"
  c(plants,markets) = f * d(plants,markets) / 1000 ;

* Model section

Positive Variable x(i,j) "shipment quantities in cases";
Variable z "total transportation costs in thousands of dollars";

Equations cost "define objective function"
  supply(i) "observe supply limit at plant i"
  demand(j) "satisfy demand at market j";

  cost .. z =e= sum((plants,markets), c(plants,markets) * x(plants,markets)) ;
  supply(plants) .. sum(markets, x(plants,markets)) =l= a(plants) ;
  demand(markets) .. sum(plants, x(plants,markets)) =g= b(markets) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Observe that we introduced the subsets plants and markets that contain only some of the elements of their supersets i and j. Note that all tables, parameters and variables are defined with the supersets, the equations are declared over the supersets, but defined over the subsets and the calculation of the parameter c is also restricted to the subsets. Hence the model is restricted to the elements of the subsets. However, it is easy to change the restricted model back to the full model by removing the asterisks indicating a comment line:
plants(i) = yes; markets(j) = yes;

Observe that the sets plants and markets are now dynamic sets. Note that this assignment could be inserted anywhere in the code. Thus, introducing strategic subsets may be combined with isolating problem areas, as detailed in section Isolating Problem Areas through Saving and Restarting above.

Introducing strategic subsets has proven to be an effective way of maintaining a small data set with little effort. Users only have to choose elements that are representative for model development and debugging from the full sets.

4.15.2.3.3 Reducing Data  Recall that GAMS skips cases where data items equal zero. Thus a large model may be reduced by temporarily removing data from data sets by simply setting items to zero. Consider the following example:

```gams
Sets o 'origin' / o1*o100 / d 'destination' / d1*d100 /;
Parameter dist(o,d) 'distance';
dist(o,d) = 120 + 50*ord(d) - 0.5*ord(o);
Sets so(o) 'small set of origins for testing' / o4, o47, o91 / sd(d) 'small set of destinations' / d3, d44, d99 /;
dist(o,d) $ (not (so(o) and sd(d))) = 0;
Parameter cost(o,d) 'transportation cost';
cost(o,d) $ dist(o,d) = 3 + 2*dist(o,d);
display cost, dist;
```

Note that we introduced strategic subsets and used them in the logical condition of a conditional assignment to set almost all entries of the parameter dist to zero. Note further, that the assignment for the parameter cost is conditioned on nonzero entries for the distance. Now, if the model were conditioned on nonzero transportation costs, the size of the whole model would be greatly reduced.

4.15.3 Increasing Efficiency: Reducing GAMS Execution Time

GAMS can take a long time for computations and model generation. There are some signs which indicate that it may be possible to reduce the execution time, e.g., an execution time that is unexpectedly long in general or a long execution of a single line, which could be seen, if the log shows the same line number for a long time.

In this section we will discuss how to find the causes for slow program execution and how to eliminate the main causes for slow execution.

4.15.3.1 Finding the Causes for Slow Program Execution

The best strategy for discovering the causes for slow execution is a combination of the techniques discussed in section Small to Large: Aid in Development and Debugging above and the techniques that we will introduce in this section, including generating an execution profile and isolating terms in slow statements. We will also touch briefly on observing the log file and we will point out why this is not the first choice.
4.15.3.1.1 Generating an Execution Profile  The quickest way to find GAMS statements that take particularly long to execute, is generating an execution profile in the output file. The execution profile contains the individual and cumulative time required to execute the sections of the GAMS model as well as information on memory use. An execution profile is generated when the option profile is assigned a value larger than zero (zero is the default). This can be done either by setting a command line parameter or by using the option statement. We will show an example of an execution profile below. For more information on execution profiles, further examples and details on the values the option profile may take, see the detailed description here.

Consider the following example:

```gams
option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets    a / 1*22 /, b / 1*22 /, c / 1*20 /,
        d / 1*20 /, e / 1*22 /;

Parameters x(e,d,c,b,a), y, z(a,b,c,d,e);
x(e,d,c,b,a) = 10;
z(a,b,c,d,e) = x(e,d,c,b,a);
y = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a));

Variable obj;
Positive Variable var(e,b,a);

Equations objeq, r(b,c,d), q(a,b,c);

objeq.. obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a) * var(e,b,a));
r(b,c,d).. sum((a,e), var(e,b,a)) =l= sum((a,e), x(e,d,c,b,a)*z(a,b,c,d,e));
q(a,b,c).. sum((d,e), var(e,b,a)/x(e,d,c,b,a)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a)*var.l(e,b,a));
display sumofvar;
```

The listing file will contain an execution profile like this (spread over the file):

```plaintext
---- 9 Assignment x  0.374  0.374 SECS  109 MB 4259200
---- 10 Assignment z  2.231  2.605 SECS  286 MB 4259200
---- 11 Assignment y  2.324  4.929 SECS  286 MB 0
---- 23 Solve Init slow 0.000  4.961 SECS  287 MB
---- 18 Equation objeq  3.510  8.471 SECS  287 MB 1
---- 19 Equation r  3.088 11.559 SECS  464 MB 8800
---- 20 Equation q  5.741 17.300 SECS  470 MB 9680
---- 23 Solve Fini slow 0.780 18.080 SECS 470 MB
---- 26 Assignment sumofvar 2.620  2.652 SECS  287 MB 0
---- 27 Display 0.032  2.684 SECS  287 MB
---- 27 GAMS Fini  0.000  0.000 SECS  287 MB
```
The first column shows the line number in the input file of the GAMS statement that is executed. The second column reports the type of statement. For an overview of all GAMS statements, see section Classification of GAMS Statements. The next two columns give the individual time needed to execute the respective statement and the cumulative time spent so far. The memory use follows and finally, the number of assignments generated in the respective line is shown.

In addition, there is a Profile Summary at the end of the lst file showing the most expensive statements:

--- Profile Summary (19 records processed)---

<table>
<thead>
<tr>
<th>Time</th>
<th>Memory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.741</td>
<td>0.470GB</td>
<td>Equation q (9680)</td>
</tr>
<tr>
<td>3.510</td>
<td>0.287GB</td>
<td>Equation objeq (1)</td>
</tr>
<tr>
<td>3.088</td>
<td>0.464GB</td>
<td>Equation r (8800)</td>
</tr>
<tr>
<td>2.620</td>
<td>0.287GB</td>
<td>Assignment sumofvar (0)</td>
</tr>
<tr>
<td>2.324</td>
<td>0.286GB</td>
<td>Assignment y (0)</td>
</tr>
<tr>
<td>2.231</td>
<td>0.286GB</td>
<td>Assignment z (4259200)</td>
</tr>
<tr>
<td>0.780</td>
<td>0.470GB</td>
<td>Solve Fini slow (4482809)</td>
</tr>
<tr>
<td>0.374</td>
<td>0.109GB</td>
<td>Assignment x (4259200)</td>
</tr>
</tbody>
</table>
| 0.359| 0.470GB| GAMS Fini|}

This shows that the execution of the statements in line numbers 20, 18, 19, 26, 11 and 10 are the most expensive ones (in this order). One reason is an inconsistent order when sets are referenced; we will discuss this topic in section Ordering Indices Consistently below.

Note that the execution profile may contain many lines that are not informative since the execution times reported are negligible. These lines may be suppressed by using the the option profileTol to specify the minimum execution time (in seconds) that is to be included. Observe that the option profileTol is available as command line parameter and option statement.

Note further, that the command line parameter profileFile facilitates writing the profiling information to a separate file (instead of the listing file).

### 4.15.3.1.2 Isolating Terms in Slow Statements

In some cases the execution profile shows that the cause for a long execution time is connected with a very long statement. For example, the objective function in some models and some report calculations may take hundreds of lines of code and can contain many terms that are added. If such a long statement is problematic in terms of execution time, it will be necessary to deactivate parts of the code and run the program repeatedly to find the precise lines that are at the root of the problem. This can be done by using comments.

### 4.15.3.1.3 Observing the Log File

Some modelers choose to examine the log file or watch the screen during execution to find the causes for slow program execution. However, we advise against this approach for the following reasons:

- Statements that are executed slowly are easily missed and often statements are misidentified. In addition, screen watchers may be distracted and will have to repeat the process.
- GAMS line reporting can be misleading if flow control statements like if statements and loop statements are executed. For example, individual calculations in a loop are not reported to the screen. A user watching the screen would notice that the loop takes a lot of time, but there is no indication which statement within the loop is problematic. This applies to all GAMS control structures.

Therefore we recommend to use the option profile as the main tool for finding the causes for slow program execution. For details, see section Generating an Execution Profile above. In addition, see the techniques outlined in section Advice for Repairing Puzzling Nonworking Code below.
4.15.3.2 Eliminating the Main Causes for Slow Program Execution

The main reasons for a slow program execution include an inconsistent index order when sets are referenced and taking irrelevant cases into consideration. In this section we will give some guidance on how to eliminate these causes, and also point to problems due to the scaling of a model which could cause an unnecessarily long execution time for the solver.

4.15.3.2.1 Ordering Indices Consistently

GAMS employs a sparse matrix data storage scheme. For example, consider the parameter \( p(a,b,c) \). Assume that the set \( a \) has \( k \) elements, the set \( b \) has \( n \) elements and \( c \) has \( m \) elements. Then the entries for \( p \) are stored in the following order:

\[
\begin{align*}
  a_1 b_1 c_1 \\
  a_1 b_1 c_2 \\
  \vdots \\
  a_1 b_1 c_m \\
  a_1 b_2 c_1 \\
  \vdots \\
  a_1 b_2 c_m \\
  \vdots \\
  a_1 b_n c_m \\
  a_2 b_1 c_1 \\
  \vdots \\
  a_k b_n c_m
\end{align*}
\]

Note that it is a systematic order where the last entry varies the fastest and the first the slowest. Observe that GAMS will withdraw entries from memory fastest if they are referenced in an order consistent with the storage order. Thus, in the following example, the first assignment statement will be processed faster than the second assignment statement.

\[
x(a,b,c) = p(a,b,c); \\
y(b,c,a) = p(a,b,c);
\]

Note

GAMS will execute a program fastest if the sets are always referenced in the same order in definitions, assignments and equations.

The example that follows illustrates this principle. First we will solve a program where the indices appear in an arbitrary order and we will record the output generated by setting the option profile to 1. Then we will reformulate the program so that the indices will always appear in an alphabetical order and solve it again, recording the profile output. In the final step, we will compare the execution times of the two runs. We will use the example introduced above.

Note

That the indices in the parameters and equations appear in a random order. Here is the profile from the six most expensive statements again:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><strong>Assignment</strong> z</td>
<td>2.231</td>
<td>2.605 SECS</td>
</tr>
<tr>
<td>11</td>
<td><strong>Assignment</strong> y</td>
<td>2.324</td>
<td>4.929 SECS</td>
</tr>
<tr>
<td>18</td>
<td><strong>Equation</strong> objeq</td>
<td>3.510</td>
<td>8.471 SECS</td>
</tr>
<tr>
<td>19</td>
<td><strong>Equation</strong> r</td>
<td>3.088</td>
<td>11.559 SECS</td>
</tr>
<tr>
<td>20</td>
<td><strong>Equation</strong> q</td>
<td>5.741</td>
<td>17.300 SECS</td>
</tr>
<tr>
<td>26</td>
<td><strong>Assignment</strong> sumofvar</td>
<td>2.620</td>
<td>2.652 SECS</td>
</tr>
</tbody>
</table>
In the next step we reformulate the program such that the indices always appear in the same order. For example, we define the parameter $x$ as $x(a,b,c,d,e)$ instead of $x(e,d,c,b,a)$. Here is the complete rewritten model:

```plaintext
option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets a / 1*22 /, b / 1*22 /, c / 1*20 /, d / 1*20 /, e / 1*22 /;

Parameters x(a,b,c,d,e), y, z(a,b,c,d,e);

x(a,b,c,d,e) = 10;
z(a,b,c,d,e) = x(a,b,c,d,e);
y = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e));

Variable obj;
Positive Variable var(a,b,e);

Equations objeq, r(b,c,d), q(a,b,c);

objeq..   obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e) * var(a,b,e));
r(b,c,d).. sum((a,e), var(a,b,e)) =l= sum((a,e), x(a,b,c,d,e)*z(a,b,c,d,e));
q(a,b,c).. sum((d,e), var(a,b,e)/x(a,b,c,d,e)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e)*var.l(a,b,e));
display sumofvar;
```

After running the modified program, the profile for expensive statements looks like this:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>SECS</th>
<th>MB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Assignment z</td>
<td>0.593</td>
<td>0.983</td>
<td>215</td>
<td>4259200</td>
</tr>
<tr>
<td>11</td>
<td>Assignment y</td>
<td>0.671</td>
<td>1.654</td>
<td>215</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>Equation objeq</td>
<td>1.778</td>
<td>3.432</td>
<td>215</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>Equation r</td>
<td>2.215</td>
<td>5.647</td>
<td>392</td>
<td>8800</td>
</tr>
<tr>
<td>20</td>
<td>Equation q</td>
<td>1.763</td>
<td>7.410</td>
<td>398</td>
<td>9680</td>
</tr>
<tr>
<td>26</td>
<td>Assignment sumofvar</td>
<td>0.952</td>
<td>0.983</td>
<td>216</td>
<td>0</td>
</tr>
</tbody>
</table>

Observe that executing for example the assignment to $z$ took just 0.593 seconds compared to 2.231 seconds in the first run. Substantial percentage reductions were achieved in all time consuming cases by consistently referencing the sets in the same order.

### 4.15.3.2.2 Replace loops with assignments

The following statement assigns a constant value to a parameter.

```plaintext
loop((i,j,k),p(i,j,k)=2;)
```

The following assignment is preferred instead.

```plaintext
p(i,j,k)=2;
```
4.15.3.2.3 Restricting Assignments and Equations to Relevant Cases

4.15.3.2.3.1 Assignments
Assume that we have a set of cities with different production capacities and demands for various products. We want to know the maximum transportation cost (which depends on the distance, the amount shipped and a fixed factor) from each city to all others. This cost can be calculated in the following way:

Sets c "cities" / c1*c800 /  
  p "products" / p1*p10 /;  
Alias (c,cc);

Parameter capacity(c,p) "Production capacity for product p in city c"  
  demand(c,p) "Demand for product p in city c"  
  distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
  capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(150,250);  
  demand(c,p)$(uniform(0,1)<0.025) = uniformInt(50,150);  
  distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";
  
  maxCost(c,cc) = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);

The performance profile will tell us something like this:

--- 16 Assignment maxCost 0.265 0.436 SECS 19 MB 8756

Since we know, that the parameter \texttt{maxCost} will be zero for a pair of cities if there is no product with production capacity in the first city and demand in the second one, we could reduce the execution time for the last assignment:

Sets c "cities" / c1*c800 /  
  p "products" / p1*p10 /;  
Alias (c,cc);

Parameter capacity(c,p) "Production capacity for product p in city c"  
  demand(c,p) "Demand for product p in city c"  
  distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
  capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(50,150);  
  demand(c,p)$(uniform(0,1)<0.025) = uniformInt(50,150);  
  distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";
  
  maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))  
    = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);

So we did not do the calculation of \texttt{maxCost} if we knew before, that it must be zero anyway. This results in a reduced runtime:

--- 17 Assignment maxCost 0.031 0.187 SECS 19 MB 8756
Note

To restrict computations in assignment to the relevant cases, we recommend using dollar conditions and filtering sets. These concepts are introduced and discussed in detail in chapter Conditional Expressions, Assignments and Equations.

For more examples, see sections Conditional Assignments and Conditional Indexed Operations.

4.15.3.2.3.2 Variables and Equations

Like assignments, variables and equations need to be restricted to relevant cases to avoid unnecessary inefficiencies. Dollar conditions and filtering sets may be used over the domain of definition as well as in the body of an equation.

Let's extend the assignment example from the previous paragraph and use the generated data in a transportation model:

```plaintext
Sets
c "cities" / c1*c800 /
p "products" / p1*p10 /;
Alias (c,cc);
Parameter capacity(c,p) "Production capacity for product p in city c"
  demand(c,p) "Demand for product p in city c"
distance(c,cc) "Distance between two cities";
*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(150,250);
demand(c,p)$(uniform(0,1)<0.025) = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);
Parameter shipCost(c,cc) "Transportation costs between two cities per case"
  maxCost(c,cc) "Maximum transportation costs between two cities"
  = distance(c,cc)*90;
  maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))
  = sum(p, min(capacity(c,p), demand(cc,p))*shipCost(c,cc));
Variables
  x(c,cc,p) "shipment quantities in cases"
  z "total transportation costs in thousands of dollars";
Positive Variable x;
Equations
cost "define objective function"
supply(c,p) "observe supply limit at plant i"
dem(cc,p) "satisfy demand at market j";
cost.. z =e= sum((c,cc,p), shipCost(c,cc)*x(c,cc,p)) ;
supply(c,p).. sum(cc, x(c,cc,p)) =l= capacity(c,p) ;
dem(cc,p).. sum(c, x(c,cc,p)) =g= demand(cc,p) ;
Model transport /all/ ;
Solve transport using lp minimizing z ;
```

The Profile Summary tells us, that the equations are rather expensive to generate and also the reading of the solution takes some time because of the size of the model:
However, as in the previous example, we know, that a product \( p \) won’t be shipped from city \( c \) to city \( cc \) if there is either no production capacity in the first city or no demand in the second one. So we could reduce the size of our model by not generating variables and equations from which we know, that they are irrelevant for the solution. Here is a improved formulations of the equations:

\[
\begin{align*}
\text{cost}.. & \quad z = \sum((c,cc,p)$(capacity(c,p)*demand(cc,p)),\ shipCost(c,cc)*x(c,cc,p)) \\
\text{supply(c,p)$capacity(c,p)}.. & \quad \sum(cc$demand(cc,p), \ x(c,cc,p)) = l= capacity(c,p) \\
\text{dem(cc,p)$demand(cc,p)..} & \quad \sum(c$capacity(c,p), \ x(c,cc,p)) = g= demand(cc,p)
\end{align*}
\]

This decreases the size of the model and thus the execution time to generate the model and load the solution significantly:

---- Profile Summary (18 records processed)
  0.031  0.035GB  33 Equation  cost (1)  
  0.031  0.034GB  39 Solve Read transport  
  0.016  0.035GB  34 Equation  supply (380)  

Note that the equation \text{dem} does not even show up in the summary anymore since its generation was done quickly.

For more details on conditions in equations, see section Conditional Equations.

4.15.3.2.4 Keep the model well scaled  Model solutions within GAMS frequently require manipulation of large matrices and many computations. The heart of most solvers includes many numerical procedures such as a sparse matrix inverter and sets of convergence and infeasibility tolerances. Numerical problems often arise within such procedures. Poorly scaled models can cause excessive time to be taken in solving or can cause the solver to fail. GAMS can assist the user to formulate a well scaled model. Details about this can be found in the sections Model Scaling - The Scale Option and Scaling Variables and Equations.

4.15.3.3 Other Approaches

In addition to the techniques discussed in section Eliminating the Main Causes for Slow Program Execution above, the following approaches may help to reduce the time needed for program execution:

- Trying another appropriate solver.
- Reformulating the model. This may yield particularly good results, if the model is reformulated in such a way that another model type is used, that is easier to solve or for which more advanced solver technology is available.
- Using starting points for NLP models, as discussed in section Specifying Initial Values.
- Trading memory for time.
We conclude the discussion of this topic with an example that demonstrates how memory may be traded for time. If an extensive calculation is repeated many times in a model, it may be possible to restructure the code so that the calculation is performed only once, then the result is saved and accessed later. Consider the following equation:

\[
\text{obj. } z = e = \text{sum}[(i,j,k,l), \ a(i,j,k,l) * \text{sum}(m, u(m,i))];
\]

The execution time may be substantially reduced by defining a new parameter, say \( p \), for the second sum and using this parameter in the equation:

\[
\text{Parameter } p(i);
\]

\[
p(i) = \text{sum}(m, u(m,i));
\]

\[
\ldots
\]

\[
\text{obj. } z = e = \text{sum}[(i,j,k,l), \ a(i,j,k,l) * p(i)];
\]

There is only one caveat: Users need to carefully consider whether the input data, here \( u(m,i) \), is modified between the assignment for the new parameter \( p \) and the equation where \( p \) is used. If \( u \) is updated, then the assignment statement needs to be repeated, otherwise the data that enters the equation will not be current.

### 4.15.4 Increasing Efficiency: Reducing Memory Use

Besides slow program execution, excessive memory use may be of concern for modelers. In this section we will present some approaches on how to find the causes for extraordinary memory use and give some advice on eliminating the main causes for it.

#### 4.15.4.1 Finding the Causes for Excessive Memory Use

The main techniques for finding the causes for excessive memory use are the same as those for finding the causes for slow program execution. We discussed these techniques in section Finding the Causes for Slow Program Execution above.

In addition, the option \texttt{dmpUserSym} is useful in this context. GAMS will report the number of records stored for each symbol at the point in the program where the option \texttt{dmpUserSym} is inserted together with some rough memory estimate.

Consider the following example:

\[
\text{Sets i /1*5 /, j /1*5 /, k /1*5 /, l /1*5 /, m /1*5 /, n /1*5 /, o /1*5 /;}
\]

\[
\text{Parameters y(i,j,k,l,m,n,o)}
\]

\[
q(i,j,k);
\]

\[
\text{Variables x(i,j,k,l,m,n,o)}
\]

\[
f(i,j,k)
\]

\[
\text{obj;}
\]

\[
y(i,j,k,l,m,n,o) = 10;
\]

\[
q(i,j,k) = 10;
\]

\[
x.up(i,j,k,l,m,n,o) = 10;
\]

\[
x.scale(i,j,k,l,m,n,o) = 1000;
\]

\[
\text{Equations z(i,j,k,l,m,n,o)}
\]

\[
\text{Equation z;}
\]

\[
\text{z(i,j,k,l,m,n,o) = e = sum[(i,j,k,l), a(i,j,k,l) * p(i)];}
\]

\[
\text{...}
\]

\[
\text{obj.. z =e= sum[(i,j,k,l), a(i,j,k,l)*p(i)];}
\]
res(i,j,k)
ob;

ob.. obj =e= sum((i,j,k,l,m,n,o), x(i,j,k,l,m,n,o));
z(i,j,k,l,m,n,o).. x(i,j,k,l,m,n,o) =l= 8;
res(i,j,k).. f(i,j,k) =l= 7;

Model memory /all/;
option dmpUserSym;
solve memory maximizing obj using lp;

Note that an option statement with dmpUserSym was added before the solve statement. It generates the following memory dump that is included in the execution output of the listing file:

SYMBOL TABLE DUMP (USER SYMBOLS ONLY), NR ENTRIES = 16
ENTRY   ID   TYPE   DIM   LENGTH   MEMORYEST   DEFINED   ASSIGNED   DATAKNOWN
135    i    SET    1    5      0 MB    TRUE    FALSE    TRUE
136    j    SET    1    5      0 MB    TRUE    FALSE    TRUE
137    k    SET    1    5      0 MB    TRUE    FALSE    TRUE
138    l    SET    1    5      0 MB    TRUE    FALSE    TRUE
139    m    SET    1    5      0 MB    TRUE    FALSE    TRUE
140    n    SET    1    5      0 MB    TRUE    FALSE    TRUE
141    o    SET    1    5      0 MB    TRUE    FALSE    TRUE
142    y    PARAM  7    78125  3 MB   FALSE    TRUE    FALSE
143    q    PARAM  3    125   0 MB   FALSE    TRUE    FALSE
144    x    VAR    7    78125  5 MB   FALSE    TRUE    FALSE
145    f    VAR    3    0      0 MB   FALSE    TRUE    FALSE
146    obj   VAR    0    0      0 MB   FALSE    TRUE    FALSE
147    z    EQU    7    0      0 MB   FALSE    TRUE    FALSE
148    res   EQU    3    0      0 MB   FALSE    TRUE    FALSE
149    ob    EQU    0    0      0 MB   FALSE    TRUE    FALSE
150    memory MODEL  0    3    TRUE    TRUE    TRUE

END OF SYMBOL TABLE DUMP

The column ID contains the names of the symbols, the column TYPE gives the data type of the respective entry, the column DIM reports the number of indices and the column LENGTH gives the number of records that is related to memory use, which is estimated in the column MEMORYEST. Note that the other columns are not relevant for this discussion.

Observe that the rows with high counts in column LENGTH indicate symbols within the GAMS program which have large numbers of internal records that must be stored. This is associated with corresponding memory requirements. Note also that not all length counts are of equal significance. In particular, variables and equations use more memory per element than parameters, since they have bounds, levels, marginals and scales that are associated with them. Parameters use more memory per element than sets, since sets may need just one indicator for yes or no. However, the explanatory text for set elements might increase the memory requirements for set elements.

Nevertheless, users may use this report to identify items with many records and verify that all of them are actually needed. For more details, see section Eliminating the Main Causes for Excessive Memory Use below.

4.15.4.2 Eliminating the Main Causes for Excessive Memory Use

As detailed in section Eliminating the Main Causes for Slow Program Execution above, the main causes for a slow program execution include an inconsistent index order when sets are referenced and taking irrelevant cases into consideration. These programming habits also tend to cause excessive memory use. In this section we will give some advice on avoiding memory traps and show how the memory may be cleared of data that is no longer needed.
4.15 Finding and Fixing Execution Errors and Performance Problems

4.15.4.2.1 Avoiding Memory Traps  Users may inadvertently use a lot of memory if they import data from a database with long explanatory text for sets or set elements. In addition, setting variable attributes for scaling or bounds may be problematic. Consider the following example:

\[
\begin{align*}
\text{x.scale}(i,j,k,l,m) &= 100; \\
\text{x.lo}(i,j,k,l,m) &= 10; \\
\text{x.up}(i,j,k,l,m) &= 77;
\end{align*}
\]

These assignments will probably set many more values than are relevant for a particular problem. Therefore we recommend to carefully consider which label combinations are actually necessary and to restrict the assignments to these cases by the use of dollar conditions or filtering sets. For more information, see section Conditional Assignments.

4.15.4.2.2 Clearing Memory of Unneeded Data  Sometimes a lot of memory space is used for data that is needed at some point, but not later. Consider the following simple example:

\[
\begin{align*}
\text{set } i &\ /1*1000/ \\
&\ j\ /1*1000/; \\
\text{parameter}\& \text{ distance}(i,j) \\
&\ \text{cost}(i,j); \\
\text{distance}(i,j) &= 100+\text{ord}(i)+\text{ord}(j); \\
\text{cost}(i,j) &= 4+8*\text{dist}(i,j);
\end{align*}
\]

Assume that the parameter distance is used only here, but nowhere else in the program. Therefore users may wish to free the memory space occupied by the data connected with distance. The option clear may be used to achieve this:

\[
\text{option clear = distance;}
\]

This will reset all entries in the matrix associated with distance to zero.

Alternatively, an identifier that is no longer needed could be reset to its default value(s) with an assignment statement. In the example above, we could write:

\[
\text{distance}(i,j) = 0;
\]

This statement will have the same effect as the option statement. The advantage of the option statements is that they offer a more compact alternative that is particularly useful if equations or variables are to be cleared and multiple equation attributes or variable attributes are affected.

Note that the dollar control options $clear and $kill may also be used to free memory. These are compile time directives, which have a similar effect on the memory consumption but have different side effects: while $clear will reset the values to their defaults, $kill will completely remove the identifier from the program. Hence an identifier that was ”killed” may be used later in another declaration and definition statement. For example, the following code snippet is legal:

\[
\text{Set i /1, 2 ,3/;}
\text{$kill i}$
\text{Set i /a, b, c/;}
\]

With $clear instead of $kill this would cause a compilation error.
4.15.4.3 Setting Memory Limits with HEAPLIMIT

In a server environment and in other cases (e.g. to avoid the use of virtual memory) the amount of memory a GAMS run is allowed to use may have to be limited. The command line parameter heapLimit serves this purpose: the amount of memory for GAMS compilation and execution is limited to a specified number (in MB). If the data storage exceeds this limit, the job will be terminated with return code 10, out of memory. In addition, the function heapLimit may be used to interrogate the current limit and to reset it.

Note that limiting memory use for solver execution is not possible from within the GAMS program. However, some solvers like the NLP solver CONOPT have their own heapLimit option which ensures that the solver will not use more dynamic memory than specified.

4.15.4.4 Advice for Repairing Puzzling Nonworking Code

Assume a GAMS run was terminated and we cannot get a profile output (e.g. because GAMS ran out of memory and crashes). A memory overrun error causes the operating system buffer handling procedures to generally lose the last few lines of profile information when the job malfunctioned. How do we find the problem in this case?

We recommend to use the techniques outlined in section Modeling Techniques above. In addition, successively deactivating code in search for the last GAMS statement that worked will help in most cases. This can be done by using comments. If at some point the run terminates properly, the user will slowly activate parts of the last statements that were deactivated until the code performance will get worse again. By iteratively activating and deactivating terms, the precise problematic terms may be found. The save and restart feature could also be used to save the results until a certain statement and then to execute only the statements that are suspected to be problematic.

4.16 Comparative Analyses with GAMS

Once a model is completed, it is almost always used to investigate alternative scenarios where the analyst compares the results of various scenario assumptions. In this tutorial we will show how such comparative analyses (also called sensitivity analyses) are done with GAMS. We will first demonstrate an easy approach, where we will manually change input parameters, use repeated solves and generate reports. In a second step, we will introduce another approach, where a loop structure will be used to automatically cycle through the scenarios. We recommend to read the sections on the manual approach first, since the sections on the automated approach build on code blocks developed in the early sections.

4.16.1 Manual Approach

Suppose we wish to do a comparative analysis by altering some input data in a model. We will use as an example the farm profit-maximizing model farmcomp.gms. The following vector of prices for primary commodities is a part of the input data:

Parameter price(primary) 'prices for products in USD'
   / corn 2.20, soybeans 5.00, beef 0.50 /;
We will use these data as a base case and compare it with two alternative scenarios: in the first scenario we will change the price of beef to $0.70 and in the second scenario we will change the price of corn to $2.70.

The GAMS file farmrep.gms is related to our example model. It contains only calculations for report writing and may be included with the dollar control option $include. It will generate a report based on the solution of the last solve that was executed in the GAMS program farmcomp.gms. The report consists of several tables. We will focus on the table Farm Summary that is associated with the parameter summary. The relevant code is given below:

Set alli 'allitems'
/ corn, soybeans, beef, cattle,
   water, cropland, pastureland,
   fertilizer, seed, othercost, veterinary, supplement,
   "April labor", "May labor", "summer labor", "Sept labor", "Oct labor",
   cattlefeed, total /;

Set measures 'output measures'
/ "Net Income", "Land Use", "Dry Cropping", "Irr Cropping",
 "Livestock", "Resource Value", "Product Value" /;

Parameter summary(alli,measures) 'Farm Summary';

Note that the table for the parameter summary will contain rows for the commodities which are elements of the set alli and columns for all elements of the set measures.

We will use a third GAMS file, mancomp.gms, for our comparative analysis. The code of this third GAMS file follows:

$include farmcomp.gms
display price;
$include farmrep.gms
price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms
price("corn") = 2.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms

Note that this code first solves the original model that also contains the set definitions for the report, displays the initial prices and generates a report. In a second step the price for beef is changed to $0.70, the modified model is solved, the prices for the first alternative scenario are displayed and a report is generated. In a third step the price for corn is changed to $2.70, the model is solved again, the prices for the second alternative scenario are displayed and a third report is generated. Note that in the second alternative scenario (the third solve) the beef price is $0.70, since it was not reset to base levels after the second run.

There will be three tables associated with the parameter summary in the listing file, one for each solve. The first table reports the results associated with the base case:
### 279 PARAMETER summary Farm Summary

<table>
<thead>
<tr>
<th>Net Income</th>
<th>Land use</th>
<th>Dry Cropp$^*$</th>
<th>Irr Cropp$^*$</th>
<th>Livestock Resource $^*$</th>
<th>Product V$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corn</td>
<td>20.00</td>
<td>200.00</td>
<td></td>
<td></td>
<td>2.20</td>
</tr>
<tr>
<td>Soybeans</td>
<td>480.00</td>
<td></td>
<td></td>
<td></td>
<td>5.00</td>
</tr>
<tr>
<td>Beef cattle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.50</td>
</tr>
<tr>
<td>Water</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>615.79</td>
</tr>
<tr>
<td>Cropland</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16.83</td>
</tr>
<tr>
<td>Pastureland</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128.49</td>
</tr>
<tr>
<td>April Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84.26</td>
</tr>
<tr>
<td>May Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32.34</td>
</tr>
<tr>
<td>Oct Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>27.01</td>
</tr>
<tr>
<td>Cattlefeed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.50</td>
</tr>
<tr>
<td>Total</td>
<td>162685.05</td>
<td>500.00</td>
<td>200.00</td>
<td></td>
<td>4.71</td>
</tr>
</tbody>
</table>

The second table reports the results from the first alternative scenario where the price for beef was changed to $0.70:

### 351 PARAMETER summary Farm Summary

<table>
<thead>
<tr>
<th>Net Income</th>
<th>Land use</th>
<th>Dry Cropp$^*$</th>
<th>Irr Cropp$^*$</th>
<th>Livestock Resource $^*$</th>
<th>Product V$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corn</td>
<td>22.84</td>
<td>160.85</td>
<td></td>
<td></td>
<td>2.34</td>
</tr>
<tr>
<td>Soybeans</td>
<td>489.86</td>
<td></td>
<td></td>
<td></td>
<td>5.00</td>
</tr>
<tr>
<td>Beef cattle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.70</td>
</tr>
<tr>
<td>Cropland</td>
<td>673.55</td>
<td></td>
<td></td>
<td></td>
<td>866.67</td>
</tr>
<tr>
<td>Pastureland</td>
<td>130.00</td>
<td></td>
<td></td>
<td></td>
<td>1456.90</td>
</tr>
<tr>
<td>April Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82.29</td>
</tr>
<tr>
<td>May Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80.53</td>
</tr>
<tr>
<td>Sept Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>53.57</td>
</tr>
<tr>
<td>Oct Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>46.21</td>
</tr>
<tr>
<td>Cattlefeed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.89</td>
</tr>
<tr>
<td>Total</td>
<td>373686.10</td>
<td>512.70</td>
<td>160.85</td>
<td></td>
<td>4.89</td>
</tr>
</tbody>
</table>

And the third table reports the results from the second alternative scenario where the price for corn was changed to $2.70 and the price for beef stayed at $0.70:

### 423 PARAMETER summary Farm Summary

<table>
<thead>
<tr>
<th>Net Income</th>
<th>Land use</th>
<th>Dry Cropp$^*$</th>
<th>Irr Cropp$^*$</th>
<th>Livestock Resource $^*$</th>
<th>Product V$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corn</td>
<td>31.98</td>
<td>200.00</td>
<td></td>
<td></td>
<td>2.70</td>
</tr>
<tr>
<td>Soybeans</td>
<td>410.24</td>
<td></td>
<td></td>
<td></td>
<td>5.00</td>
</tr>
<tr>
<td>Beef cattle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.70</td>
</tr>
<tr>
<td>Water</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>866.67</td>
</tr>
<tr>
<td>Cropland</td>
<td>642.22</td>
<td></td>
<td></td>
<td></td>
<td>15.99</td>
</tr>
<tr>
<td>Pastureland</td>
<td>130.00</td>
<td></td>
<td></td>
<td></td>
<td>1316.09</td>
</tr>
<tr>
<td>April Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>61.39</td>
</tr>
<tr>
<td>May Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>61.72</td>
</tr>
<tr>
<td>Sept Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84.92</td>
</tr>
<tr>
<td>Oct Labor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>87.21</td>
</tr>
<tr>
<td>Cattlefeed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.36</td>
</tr>
<tr>
<td>Total</td>
<td>375839.30</td>
<td>442.22</td>
<td>200.00</td>
<td></td>
<td>5.36</td>
</tr>
</tbody>
</table>
This quick way to do a comparative analysis has the following drawbacks:

- The relevant output is spread over more than 140 lines.
- There is no cross-scenario report.
- The price of beef in the second alternative scenario is problematic, since it was not automatically reset to the original base price.
- The handling of solves and report writing is repetitive.

The first two issues will be addressed in section Writing Cross-Scenario Reports, a solution for the third issue will be given in section Resetting Data to Base Levels and an alternative approach that will resolve the last issue is presented in section An Automated Approach - Avoiding Repeated Work.

4.16.1.1 Writing Cross-Scenario Reports

We will generate a cross-scenario report by introducing two new sets and a new parameter in a revised version of the third GAMS file, `mancombp.gms`:

```gams
Set scenarios / base, beefp, beefcorn /;
Set ordr / "Scenario Setup", "Scenario Results"/;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);
```

Note that the set `scenarios` contains the base case and the two alternative scenarios and the set `ordr` introduces places to save the assumptions and results of the different runs. The new parameter `savsumm` is similar to the parameter `summary` introduced above, but it has two additional dimensions. Observe that the first assignment copies the current setup of the price vector and the second assignment copies the results that are stored in the parameter `summary`.

The full code follows:

```gams
$include farmcomp.gms
$include farmrep.gms
Set ordr / "Scenario Setup", "Scenario Results" /;
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);

price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price ;
$include farmrep.gms
savsumm("Scenario Setup","price",primary,"beefp") = price(primary);
savsumm("Scenario Results",measures,alli,"beefp") = summary(alli,measures);

price("corn") = 2.70;
display price ;
solve farm using LP maximizing netincome;
$include Farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn") = price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn") = summary(alli,measures);

option savsumm:2:3:1;
display savsumm;
```
Observe that the last index in the assignments for `savsumm` is "base" after the first solve, "beefp" after the second solve and "beefcorn" after the third solve. Note that option statement in the penultimate line of the code customizes the output generated by the display statement that follows. For details see section Local Display Control. The listing file will contain the following output:

```
----- 436 PARAMETER savsumm Comparative Farm Summary

<table>
<thead>
<tr>
<th>Scenario Setup .price .Corn</th>
<th>base</th>
<th>beefp</th>
<th>beefcorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario Setup .price .Soybeans</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Scenario Setup .price .Beef</td>
<td>0.50</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>Scenario Results.Net Income .Total</td>
<td>162685.05</td>
<td>373686.10</td>
<td>375839.30</td>
</tr>
<tr>
<td>Scenario Results.Land use .Cropland</td>
<td>700.00</td>
<td>673.55</td>
<td>642.22</td>
</tr>
<tr>
<td>Scenario Results.Land use .Pastureland</td>
<td>130.00</td>
<td>130.00</td>
<td>130.00</td>
</tr>
<tr>
<td>Scenario Results.Dry Cropping .Corn</td>
<td>20.00</td>
<td>22.84</td>
<td>31.98</td>
</tr>
<tr>
<td>Scenario Results.Dry Cropping .Soybeans</td>
<td>480.00</td>
<td>489.86</td>
<td>410.24</td>
</tr>
<tr>
<td>Scenario Results.Dry Cropping .Total</td>
<td>500.00</td>
<td>512.70</td>
<td>442.22</td>
</tr>
<tr>
<td>Scenario Results.Irr Cropping .Corn</td>
<td>200.00</td>
<td>160.85</td>
<td>200.00</td>
</tr>
<tr>
<td>Scenario Results.Irr Cropping .Total</td>
<td>200.00</td>
<td>160.85</td>
<td>200.00</td>
</tr>
<tr>
<td>Scenario Results.Livestock .cattle</td>
<td>615.79</td>
<td>866.67</td>
<td>866.67</td>
</tr>
<tr>
<td>Scenario Results.Resource Value.Cropland</td>
<td>128.49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario Results.Resource Value.Pastureland</td>
<td>84.26</td>
<td>1456.90</td>
<td>1316.09</td>
</tr>
<tr>
<td>Scenario Results.Resource Value.April Labor</td>
<td>32.34</td>
<td>82.29</td>
<td>61.39</td>
</tr>
<tr>
<td>Scenario Results.Resource Value.May Labor</td>
<td>27.01</td>
<td>80.53</td>
<td>61.72</td>
</tr>
<tr>
<td>Scenario Results.Resource Value.Sept Labor</td>
<td>32.34</td>
<td>82.29</td>
<td>61.39</td>
</tr>
<tr>
<td>Scenario Results.Resource Value.Oct Labor</td>
<td>11.50</td>
<td>46.21</td>
<td>87.21</td>
</tr>
<tr>
<td>Scenario Results.Product Value .Corn</td>
<td>2.20</td>
<td>2.34</td>
<td>2.70</td>
</tr>
<tr>
<td>Scenario Results.Product Value .Soybeans</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Scenario Results.Product Value .Beef</td>
<td>0.50</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>Scenario Results.Product Value .Cattlefeed</td>
<td>4.71</td>
<td>4.89</td>
<td>5.36</td>
</tr>
</tbody>
</table>
```

In this cross-scenario report all output is in one table and it is easy to compare the base case with the two alternative scenarios.

We could also add percentage change calculations by introducing a further parameter, `savsummp`:

```
Parameter savsummp(ordr,*,alli,scenarios) 'Comparative Farm Summary (percent chg)';

savsummp(ordr,measures,alli,scenarios)$savsummp(ordr,measures,alli,"base") =
  round{ [savsummp(ordr,measures,alli,scenarios) - savsummp(ordr,measures,alli,"base")]*100 /
  savsummp(ordr,measures,alli,"base"),1 }

savsummp(ordr,measures,alli,scenarios)$[(savsummp(ordr,measures,alli,"base") = 0) and (savsummp(ordr,measures,alli,scenarios) <> 0)] = na;

option savsummp:1:3:1;
display savsummp;
```

Note that both assignment statements are conditional assignments. The first assignment computes percentage changes rounded to one decimal place and the second assignment sets the percentage change to NA if the value in the base case is zero. The output generated by the display statement follows:
4.16 Comparative Analyses with GAMS

4.16.1.2 Resetting Data to Base Levels

In the example above the price for beef was changed to $0.70 for the first alternative scenario and it stayed at $0.70 in the second alternative scenario, since we did not reset it manually to the base level. However, in most cases users find it preferable to reset all data to base levels before a new scenario is run. This may be done by saving the base level data in a new parameter, say `saveprice`, and then resetting the data to base levels before each scenario. In the following final version of the third GAMS file, `mancompc.gms`, the levels of the commodity prices are reset before each new run:

```gams
$include farmcomp.gms
$include farmrep.gms

Parameter saveprice(alli) 'saved prices';
saveprice(alli) = price(alli);

Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);

price(alli) = saveprice(alli);
price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefp") = price(primary);
savsumm("Scenario Results",measures,alli,"beefp") = summary(alli,measures);

price(alli) = saveprice(alli);
price("corn") = 2.70;
display price;
solve farm using LP maximizing netincome;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn") = price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn") = summary(alli,measures);

option savsumm:2:3:1;
display savsumm;
```

---

### PARAMETER savsummp Comparative Farm Summary (percent chg)

| Scenario Results.Irr Cropping .Corn | -19.6 |
| Scenario Results.Irr Cropping .Total | -19.6 |
| Scenario Results.Livestock .cattle | 40.7  |
| Scenario Results.Resource Value.Water | -100.0 |
| Scenario Results.Resource Value.Cropland | -100.0 |
| Scenario Results.Resource Value.Pastureland | 1629.0 |
| Scenario Results.Resource Value.April Labor | 154.4 |
| Scenario Results.Resource Value.May Labor | 198.1 |
| Scenario Results.Resource Value.Sept Labor | NA |
| Scenario Results.Resource Value.Oct Labor | 301.8 |
| Scenario Results.Product Value .Corn | 6.4  |
| Scenario Results.Product Value .Beef | 40.0  |
| Scenario Results.Product Value .Cattlefeed | 3.9  |
In this section we demonstrated how to do a comparative analysis manually, including a cross-scenario report and resetting the data before each new scenario is run. However, there is still a lot of repetition in how the solves and reports are handled. This issue will be addressed in the next section.

4.16.2 An Automated Approach - Avoiding Repeated Work

The basic structure of a comparative analysis that avoids repetitive instructions for solves and report writing is outlined in the Figure below. Note that the first three boxes represent preparatory steps that are the usual parts of a GAMS program: the initial data is set up, the model is defined and solved. The comparative model analysis begins with the box labeled "Step 1". In Step 1 names for the scenarios are introduced and the scenario data is defined. In Step 2 the base data that will be changed during the scenario runs are saved in a new parameter. We will use this parameter later to restore the data to their base levels before each new scenario run. Starting with Step 3 we enter a loop, where the looping set is the set of scenarios introduced in Step 1. As usual, the statements in the loop are executed for each scenario to be analyzed. The first statement restores the data to their base levels (Step 3). Thus we will always start with the same data. In Step 4 the data and the model is updated for the current scenario and in Step 5 the model solved. In Step 6 a report for the individual scenarios is generated. In Step 7 parameters for cross-scenario comparative reports are saved. In Step 8 we check if more scenarios are to be solved and if this is the case, we return to repeat Steps 3-8 until all scenarios are completed. Finally, we display a comparative report that presents the information saved across scenarios.
This flow chart is implemented in the file `compare.gms` that is given below. Note that it is based on the farm profit-maximizing example discussed in section Manual Approach above. All sets and parameters were introduced in the sections above, only the table `scenprice` in the implementation of Step 1 is new.

*Step 1 - Setup scenarios
Set ordr / "Scenario Setup", "Scenario Results" /;
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
Table scenprice(primary,scenarios) 'price changes by scenario'
   base   beefp   beefcorn
   corn   2.70
   soybeans
   beef   0.70

*Step 2 - Save data
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);
Step 3 - Reset data to base level
loop(scenarios,
  price(primary) = savprice(primary);
)

Step 4 - Change data to levels needed in scenario
  price(primary)$scenprice(primary,scenarios) = scenprice(primary,scenarios);
  display price;

Step 5 - Solve model
  solve farm using LP maximizing netincome;

Step 6 - Single scenario report writing
  $include farmrep.gms

Step 7 - Cross-scenario report writing
  savsumm("Scenario Setup","price",primary,scenarios) = price(primary);
  savsumm("Scenario Results",measures,alli,scenarios) = summary(alli,measures);

Step 8 - End of loop
);

Step 9 - Compute and display final results
option savsumm:2:3:1;
  display savsumm;

Note that the main feature that facilitates this automatic approach is a loop statement. Loop statements are introduced and discussed in section The Loop Statement. Note further, that in Step 4 we used a conditional assignment to restrict the change of data to those primary commodities that have nonzero entries for the respective scenario in parameter scenprice. For details on conditional assignments, see section Conditional Assignments.

4.16.2.1 Adding A Scenario

Given the implementation of an automated comparative analysis above, it is easy to add a new scenario. Only two small modifications are needed: a name for the new scenario has to be added to the set scenarios and the respective data has to be added to the table scenprice. Both changes are in Step 1. The lines of the respective code are given below:

Set scenarios / base, beefp, beefcorn, new/;
Table scenprice(primary,scenarios) 'price alterations by scenario'
  base  beefp  beefcorn  new
  corn  2.70
  soybeans  4.32  0.70

Note that we added a third alternative scenario called new, where the price of soybeans is changed to $4.32. Note further, that the remainder of the code is not changed. Once the new scenario is added to the set scenarios and the respective data is specified in the table, it will enter the loop and thus all the statements in the loop will be executed for the new scenario.

Observe that the new scenario above was similar to the other scenarios: the price for a primary commodity was modified. However, in some cases new scenarios require that other data, like resources, are changed. For example, assume that in a new scenario the landtype "cropland" is increased by 30%. Note that the available resources were defined in the parameter available(alli), where available("cropland") equals 700. To accommodate this new scenario we will add the following lines of code to Steps 1 to 4:
4.16 Comparative Analyses with GAMS

*Step 1 - Setup scenarios
Set scenarios / base, beefp, beefcorn, new /;
Table scenavailable(alli,scenarios) 'resource alterations by scenario'
  base   beefp  beefcorn  new
  cropland  1.3;

*Step 2 - Save data
Parameter saveavailable (alli) 'save available resources';
saveavailable (alli) = available (alli);

*Step 3 - Reset data to base level
loop(scenarios,
  available (alli) = saveavailable (alli);
)

*Step 4 - Change data to levels needed in scenario
available(alli)$scenavailable(alli,scenarios) =
  available(alli)*scenavailable(alli,scenarios);
display price, available;

Note that in Step 1 we added a new table, scenavailable, to specify the resource alternations by scenario. Observe that the values of scenavailable will be used as multipliers in Step 4. In Step 2 we introduced the parameter saveavailable, that will play a similar role for the available resources like the parameter savprice does for the prices of the primary commodities: the base levels are saved in this parameter and we will use it to restore the values of available to their base levels before each new scenario run. This is implemented with the new assignment in Step 3. In Step 4 the data levels of the parameter available are modified as required for the current scenario using a conditional assignment. Note the similarities to the assignment for updating the values of price in Step 4 above. Steps 5 to 9 remain unchanged. Thus we achieved an automated comparative analysis with an additional scenario where another parameter is modified by adding just a few lines of code. The full code is given in the GAMS file compareother.gms.

4.16.2.2 Changing the Structure of a Model

Many studies require modifications of the structure of a model. In GAMS, context-sensitive model structures may be implemented with dollar conditions. Dollar conditions may be used to control equations as well as specific terms.

To illustrate, we will extend the profi-maximizing farm model from the previous sections to include a conditional constraint that limits the number of cattle. This constraint will be only active if a scalar that varies with the scenarios is nonzero. Consider the lines of code that follow. Note that the set animals is a subset of the set alli, the elements of the set livemanage denote various ways of managing crops and animals, and liveprod is a positive variable for livestock production.

$include farmcomp.gms
$include farmrep.gms

Scalar cowlim 'activates cowlimit constraint' /1/;
Equation cowlimit 'conditional equation on cow limits';
cowlimit$cowlim.. sum((animals,livemanage), liveprod(animals,livemanage)) =l= 100;
Model farmcowlim /all/;

Set ordr / "Scenario setup", "Scenario Results" /;
Set scenarios / base, cowlim /;
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);
Parameter cowlims(scenarios) 'cowlimit by scenario'
loop(scenarios,
    cowlim = cowlims(scenarios);
    solve farmcowlim using LP maximizing netincome;
);

Note that in the loop the value of the scalar cowlim is updated before each scenario is run. Note further, that this value determines whether the new equation cowlimit will be active or not. The complete code is given in the GAMS file comparemod.gms.

### 4.16.3 Ranging analysis

Some users are interested in getting ranging output in the form of LP cost and right hand side ranging results. Unfortunately, the base version of GAMS does not yield such information. The user wishing such information has two alternatives. First, one may cause the model to be repeatedly solved under a set of values for the key parameter using the procedures discussed above, but this is cumbersome if a lot of parameters are involved. Second, one can use solver dependent features of GAMS that can be retrieved into a GAMS parameter. Please refer to the solver manuals (e.g. for GAMS/CPLEX: Sensitivity Analysis) for further information how to use them.

### 4.17 Good NLP Formulations

In this tutorial we offer some advice and guidance on how to set up or formulate an NLP model so that a solver will be able compute a good solution and do so quickly, reliably, and predictably. Much of this applies to other model classes allowing nonlinear functions, but for ease and simplicity of exposition we focus on the NLP case here.

A good formulation for an NLP model typically involves several things, including specifying sensible initial values, setting variable bounds, and scaling variables and equations. Other factors to consider are techniques for blocking degenerate cycling and the potential benefits of avoiding expressions in nonlinear functions. Finally we look at reformulations and approximations for discontinuous functions like \(\text{abs}\), \(\text{max}\) and \(\text{min}\) in section Reformulating DNLP Models.

#### 4.17.1 Specifying Initial Values

The variable levels and equation marginals in GAMS are typically passed to a solver to be used as the initial point. The initial values specified are especially important for NLP models for several reasons:

- Non-convex models may have multiple local solutions. Local NLP solvers search for a local solution and return it when it is found. An initial point in the neighborhood of a desired solution is more likely to return that solution.
- Initial values that (almost) satisfy many of the constraints reduce the work involved in finding a first feasible solution.
- Initial values that are close to a local optimum reduce the work required to find that local optimum, therefore reducing solution time.
- The progress of the optimization algorithm is based on good directional information, i.e on good derivatives. The derivatives in a nonlinear model depend on the current point, so an improved initial point can improve solver performance.
4.17 Good NLP Formulations

Variable levels and equation marginals are specified by setting the variable attribute .L and the equation attribute .m before solution. This is often done with assignments that occur before the solve statement, e.g.:

\[
\text{domPrice}.L(i,\text{region},t) = \text{domPrice0}(i,\text{region},t);
\]
\[
\text{flowLim}.m(\text{arcs}) = 1;
\]

Note

The default value for the variable levels and equation marginals is 0.

The variable bounds also play a role in determining the initial point passed to the NLP solver. When a solve occurs, the levels for all variables in the model are first projected onto the set defined by the variable bounds. Thus, setting a variable's lower bound to a positive value ensures that the initial value of this variable will never be zero. This is very useful, since in many cases zero is an unsuitable initial value for nonlinear variables. For example, based on the product term \(x \cdot y\), an initial value of zero for \(x\) will lead to an initial derivative value of zero wrt \(y\), so it will appear as if the function does not depend on \(y\). Variables at zero can also cause numerical difficulties with logarithms, real powers, or divisions. These difficulties occur not just at zero but also for very small values (i.e. values very close to zero) as well.

We recommend to specify as many sensible initial values for the nonlinear variables as possible. It may be desirable to initialize all variables to 1 or to the scale factor if the GAMS scaling option is used. A better alternative is to first select reasonable values for some variables that are known from context or experience to be important and then to use some of the equations of the model to compute the values for other variables. For example, consider the following equation, where \(\text{pm}, \text{pwm}\) and \(\text{er}\) are variables and \(\text{tm}\) is a parameter:

\[
\text{pmDef}(i) \ldots \text{pm}(i) = \text{pwm}(i) \cdot \text{er} \cdot (1 + \text{tm}(i));
\]

The following assignment statements use the equation to derive consistent initial values for the variable \(\text{pm}\) from sensible initial values for the variables \(\text{pwm}\) and \(\text{er}\):

\[
\text{pwm}.L(i) = 1;
\]
\[
\text{er}.L = 1;
\]
\[
\text{pm}.L(i) = \text{pwm}.L(i) \cdot \text{er}.L \cdot (1 + \text{tm}(i));
\]

It would be a mistake to assign only to \(\text{pwm}\) and \(\text{er}\) and assume that the solver will choose to adjust the variable \(\text{pm}\) to make the equation feasible: it could choose to adjust \(\text{pwm}\) or \(\text{er}\) instead. With all the assignments above made, we can be assured that the equation \text{pmDef} will be satisfied at the initial point.

Setting the initial point by loading a prior solution that has been saved via the savepoint mechanism is also an effective strategy that is very easy to implement.
4.17.2 Setting Variable Bounds

Lower and upper bounds on variables are set by assigning values to the variable attributes .lo and .up in the following way:

```plaintext
price.lo(i,region,t) = 1e-4;
flow.up(arcs) = arcCap(arcs);
```

Lower and upper bounds on variables in nonlinear models serve multiple purposes. Some bounds represent constraints based on the reality that is being modeled. For example, a certain production level must be non-negative or an arc in a network has a flow capacity of at most ten. These bounds are called *model bounds*. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and/or into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called *algorithmic bounds*. Solver performance can be improved and execution errors (see domLim and domUsd) avoided when algorithmic bounds on variables are introduced.

Model bounds are determined by the reality being modeled and do not cause any problems. However, algorithmic bounds must be carefully chosen by the modeler. We recommend to pay particular attention if a variable is the argument in \( \log(x) \), \( \log_{10}(x) \) or \( \exp(x) \) and if a variable occurs in the denominator of a division. If \( \log(x) \) or \( \log_{10}(x) \) appears in a model, where \( x \) is a variable, we recommend a lower bound of \( 1.e-3 \) for \( x \) since \( \log(x) \) gets very small as \( x \) approaches zero and is undefined for negative values of \( x \). In addition, the first derivative gets very large as \( x \) approaches zero. If \( \exp(x) \) features in a model, where \( x \) is a variable, we recommend an upper bound between 20 and 25 for \( x \). If a variable \( x \) appears in the denominator, we recommend a lower bound of \( 1.e-2 \) for \( x \), since \( 1/x \) is extremely nonlinear for small arguments. Small values for variables used with negative exponents are also not desirable. Solver performance can be improved and execution errors avoided when one introduces algorithmic bounds on variables.

Note that lower and upper bounds facilitate finding a feasible solution as most solvers will honor bounds at all times, but inequalities are not necessarily satisfied at intermediate points. A further advantage of variable bounds compared to inequalities is improved presolve performance: NLP solver preprocessors will typically incur little or no computational overhead due to variable bounds.

4.17.3 Avoiding Expressions in Nonlinear Functions

It is often useful to avoid nonlinear functions of expressions (e.g. a division by the sum of many variables). Instead, an *intermediate* variable can be used for the expression. This applies in particular if the expressions depend on many variables. Consider the following example:

```plaintext
variable x(i), y;
equation ydef;
ydef.. y =e= 1 / sum(i, x(i));
```

This example could be reformulated via the intermediate variable \( x_{sum} \) and its defining equation \( x_{sumdef} \) in the following way:

```plaintext
variable x(i), y, xsum;
equation xsumdef, ydef;
xsumdef.. xsum =e= sum(i, x(i));
ydef .. y =e= 1/xsum;
xsum.lo = 1.e-2;
```

In the equation \( ydef \), the intermediate variable \( x_{sum} \) appears in the denominator instead of the original summation. This allows us to impose a lower bound on the variable \( x_{sum} \) to avoid dividing by zero. Of course, the model will contain more rows and columns if intermediate variables are introduced, but this increase in size is offset by a decrease in complexity and, in many cases, by an increase in sparsity as well.
4.17.4 Scaling Variables and Equations

Recall that nonlinear programming algorithms use the derivatives of the objective function and the constraints to find good search directions and they use function values to determine if constraints are satisfied or not. The relative size of the derivatives and the function values is influenced by the units of measurement that are used for the variables and constraints, and will have an effect on the performance of the solver and the result computed. Therefore, a proper, consistent scaling of the model is important to the success of the solution algorithm and the quality of the answer returned.

For example, assume that two goods are equally costly: both cost $1 per kg. However, the first is specified in grams and the second in metric tons, so that their coefficients in the cost function will be vastly different: $1000 per gram and 0.001 per ton respectively. If cost is measured in $1000 units, then the coefficients will be 1 and 1.e-6 respectively. This discrepancy in size may cause the algorithm to ignore the variable with the smaller coefficient, since the coefficient is comparable to some of the zero tolerances. To avoid such problems, the units of measurements need to be carefully chosen, that is, variables and constraints need to be properly scaled.

We recommend scaling with the following goals in mind:

- Solution level values of variables should fall into a range around 1, e.g. from 0.01 to 100.
- The magnitude of the nonzero constraint marginals at solution should fall into a range around 1, e.g. from 0.01 to 100.
- The magnitude of the derivatives of the nonlinear terms (i.e. the Jacobian elements) should fall into a range around 1, e.g. from 0.01 to 100, both at the initial point and at the solution.
- The constants in the equations should have absolute values around 1, e.g. from 0.01 to 100.

Well-scaled variables are measured in appropriate units. In most cases users should select the unit of measurement for the variables such that their expected value is around unity. Of course, there will always be some variation. For example, if \( x(i) \) is the production at location \( i \), one could select the same unit of measurement for all components of \( x \), say, a value around the average capacity.

In well-scaled equations the individual terms are measured in appropriate units. After choosing units for the variables users should choose the unit of measurement for the equations such that the expected values of the individual terms are around 1. For example, if these rules are followed, material balance equations will usually have coefficients of 1 and -1.

Usually well-scaled variables and equations result in well-scaled derivatives. To check whether the derivatives are well-scaled, we recommend running the model with a positive value for the option limrow and inspecting the coefficients in the equation listing of the GAMS list file.

For more about scaling in GAMS, see section Model Scaling - The Scale Option. Note that while many solvers have internal scaling procedures, a better result can generally be achieved by a judicious choice of units by the model developer.

4.17.5 Blocking Degenerate Cycling

Most commercial linear programming solvers use a perturbation technique to avoid degenerate cycling during the solution process: they temporarily add small numbers to the right-hand sides of equations. In general, NLP solvers do not have such an internal feature. Sometimes the success and performance of an NLP solver can be enhanced by a manual perturbation formulation.

In particular, if users observe that the NLP solution process has a large number of iterations where the solver does not make significant progress in altering the objective function value, we recommend to modify
the equations in the model by replacing the value of zero on the right-hand side with a small number. This may accelerate the solution process. Assume that we have the following equation:

\[ f(x) \leq 0 \]

This could be reformulated as:

\[ f(x) \leq \delta \ast 0.001 \]

Here we set \( \delta \) to 1 if we wish to keep the addition and to zero otherwise. The value of 0.001 is just an example and needs to be adjusted based on the model context. The number should be chosen such that it does not introduce significant distortions into the problem solution. Such an addition quite frequently reduces solution time by helping the solver avoid degenerate cycling. If it is done correctly, the resulting model solution is not qualitatively different from the original model solution. Users may also first solve the model with \( \delta = 1 \) and subsequently with \( \delta = 0 \) to get rid of the effects of the small numbers.

We recommend that user avoid using the same number but instead use some systematically varying number or a random number. The technique of adding a small number on the right-hand side may also be used in problems where many equations have the same nonzero value on the right-hand side.

### 4.17.6 Reformulating DNLP Models

Nonlinear models in GAMS belong to one of the following two classes: smooth and discontinuous. Typically, all functions with endogenous arguments contained in the model are smooth functions (i.e. functions with continuous derivatives) like \( \text{sin}, \text{exp} \) and \( \text{log} \). These models can be solved using NLP. If any of the endogenous functions in the model are not smooth (i.e. are discontinuous), the model cannot be solved as an NLP: the DNLP model type must be used instead. Examples of non-smooth functions include \( \text{ceil} \) and \( \text{sign} \), where the function itself is not continuous, and \( \text{max}, \text{min}, \) and \( \text{abs} \), where the derivatives are not continuous. Typically, NLP solvers are designed to work with continuous derivatives, and much of the convergence theory behind them assumes this continuity. Discontinuous functions or derivatives may cause numerical problems, poor performance, spurious (i.e. wrong) solutions, and other issues, so they should be used with special care and only if necessary.

N.B.: to avoid a proliferation of model types, nonlinear programming is the only model type split into smooth (NLP) and nonsmooth (DNLP) variants. All other model types allowing nonlinear functions (e.g. MINLP, MCP, CNS) include both smooth and nonsmooth functions. This is not because nonsmooth functions are less problematic in these contexts. It simply became too unwieldy to maintain this distinction across all types of nonlinear models.

A powerful and effective way to model discontinuous functions is with binary variables, which results in a model of type MINLP. The model [ABSMIP] demonstrates this formulation technique for the functions \( \text{abs}, \text{min}, \text{max} \) and \( \text{sign} \). Alternatively, reformulations or approximations may be used to model discontinuous functions such that the resulting model is of type NLP. Here we offer some guidance on how to reformulate or approximate the discontinuous functions \( \text{abs}, \text{max} \) and \( \text{min} \) using only smooth functions and continuous variables and thus transform a DNLP model into an NLP. This transformation is generally more reliable than solving the original as a DNLP.

Note that some of the reformulations suggested below enlarge the feasible space. They rely on the objective function to choose a solution that is contained in the original feasible space, i.e. where the relationship defined by the nonsmooth function holds. If the objective cannot be relied on to do this, it is also possible to use one of the smooth approximations for the nonsmooth functions defined below.
4.17.6.1 Reformulating and Approximating the Function ABS

The function \(\text{abs}(x)\) returns the absolute value of the argument \(x\). If we are minimizing an absolute value, then we can split this value into its positive and negative parts (both represented by positive variables) and minimize the sum of these variables. This formulation enlarges the feasible region but the optimal solution will be the one where the sum of the positive and negative parts is equal to the absolute value.

\[
\begin{align*}
\text{variables} & \quad x, y, z; \\
\text{equations} & \quad \text{obj '1-norm'} \\
& \quad f; \\
& \quad \text{obj.. abs}(x) + \text{abs}(y) =E= z; \\
& \quad f \quad \text{sqr}(x-3) + \text{sqr}(y+5) =L= 1; \\
\text{model nonsmooth / obj, f /}; \\
\text{solve nonsmooth using dnlp min z;} \\
\text{positive variables} & \quad \text{xPlus, xMinus, yPlus, yMinus;} \\
\text{equations} & \quad \text{obj2 'smooth version of 1-norm'} \\
& \quad \text{xDef} \\
& \quad \text{yDef} \\
& \quad \text{obj2.. xPlus + xMinus + yPlus + yMinus =E= z;} \\
& \quad \text{xDef.. x =E= xPlus - xMinus;} \\
& \quad \text{yDef.. y =E= yPlus - yMinus;} \\
\text{model smooth / obj2, xDef, yDef, f /}; \\
\text{solve smooth using nlp min z;}
\end{align*}
\]

Note that the discontinuity in the derivative of the function \(\text{abs}\) has been converted into lower bounds on the new variables \(x\text{Plus, x}\text{Minus, etc: these bounds are handled routinely by any NLP solver. Note too that the feasible space is larger than before. For example, many pairs x}\text{Plus and x}\text{Minus satisfy our equation x =E= x}\text{Plus - x}\text{Minus. However, our objective ensures that one of x}\text{Plus and x}\text{Minus will be zero at the solution so that the sum of x}\text{Plus and x}\text{Minus will be the absolute value of x.}

In case the objective function does not contain a term that tries to minimize the absolute value, a smooth approximation can be used instead of the reformulation described above. This approximation should be close to the absolute value and also have smooth derivatives. Such an approximation for \(\text{abs}(f(x))\) is:

\[
\sqrt{\text{sqr}(f(x)) + \text{sqr}(\text{delta})}
\]

Here \(\text{delta}\) is a small scalar. The value of \(\text{delta}\) controls the accuracy of the approximation and the curvature around \(f(x)=0\). The approximation error is largest when \(f(x)=0\) and decreases as \(f(x)\) gets farther from zero. A value for \(\text{delta}\) ranging between 1.e-3 and 1.e-4 should be appropriate in most cases. Users could also use a larger value in an initial optimization, reduce it and solve the model again. If \(\text{delta}\) is reduced below 1.e-4, then large second order terms might lead to slow convergence or even prevent convergence. An example of this approximation used for the previous example is below:

\[
\$\text{macro MYABS(t,d)} \quad [\sqrt{\text{sqr}(t)+\text{sqr}(d)}]\n\]

\[
\text{equation obj3;}
\text{obj3.. MYABS(x,1e-4) + MYABS(y,1e-4) =E= z;} \\
\text{model approx / obj3, f /}; \\
\text{solve approx using nlp min z;}
\]
Note the use of the macro facility to encapsulate the smooth reformulation. As mentioned above, this approximation has its largest error where \( f(x) = 0 \). If it is important to get accurate values at this point, then we recommend the following alternative approximation:

\[
\sqrt{\left(f(x) + \delta\right)^2 + \left(f(x) + \delta\right)^2} - \delta
\]

Note that the only difference is the subtraction of the constant term \( \delta \). In this case, the error will equal zero at \( f(x) = 0 \) and it will increase to \(-\delta\) as \( f(x) \) moves away from zero.

### 4.17.6.2 Reformulating and Approximating the Function MAX

The function \( \text{max}(x_1, x_2, x_3, \ldots) \) returns the maximum value of the arguments, where the number of the arguments may vary. Typically, the equation

\[
t \geq \text{max}(f(x), g(y))
\]

is replaced by two inequalities:

\[
t \geq f(x) \\
t \geq g(y)
\]

Here \( x, y \) and \( t \) are variables and \( f(x) \) and \( g(y) \) are some functions depending \( x \) and \( y \) respectively. Provided the objective function has some term that tries to minimize \( t \), one of the constraints will become binding at solution and \( t \) will equal the maximum of the two terms. The extension to more than two arguments in the function \( \text{max} \) should be obvious. A simple example follows:

```plaintext
variables  
x / LO 0, L 0.2, UP [pi/2] /  
xm 'max[sin(x),cos(x)]'  
z 'objective var'  
;
equation oDef;  
oDef.. x / 100 + max[sin(x),cos(x)] =E= z;  

model nonsmooth / oDef /;  
solve nonsmooth using dnlp min z;  

equations oDef2, sinBnd, cosBnd;  
oDef2.. x / 100 + mx =E= z;  
sinBnd.. mx =G= sin(x);  
cosBnd.. mx =G= cos(x);  

model smooth / oDef2, sinBnd, cosBnd /;  
solve smooth using nlp min z;  

In case the objective function does not force the max-term to be minimized, a smooth approximation for \( \text{max}(f(x), g(y)) \) can be used, as in the following example code:

\[
\left[\frac{f(x) + g(y) + \sqrt{(f(x)-g(y))^2 + \delta^2}}{2}\right]
\]

\[
\left[\frac{f(x) + g(y) + \sqrt{(f(x)-g(y))^2 + \delta^2}}{2}\right]
\]
4.18 Data Exchange with Other Applications

```plaintext
$macro MYMAX(t1,t2,d) [0.5 * [t1 + t2 + sqrt(sqr(t1-t2) + sqr(d))] ]
equation oDef3;
oDef3.. x / 100 + MYMAX(sin(x),cos(x),1e-4) =E= z;
model approx / oDef3 /;
solve approx using nlp min z;
```

Here $delta$ is a small scalar, preferably ranging from 1.e-2 to 1.e-4. The approximation error takes its maximum of $delta/2$ when $f(x)=g(y)$ and decreases as one moves away from this point. To shift the error away from the point of discontinuity, the following approximation can be used:

$$\frac{[f(x) + g(y) + \sqrt{sqr(f(x)-g(y)) + sqr(delta))} - delta]}{2}$$

### 4.17.6.3 Reformulating and Approximating the Function MIN

The reformulation of and approximation to the $\min$ function is similar to the $\max$ case above and will not be repeated in full here. Briefly,

$$t =e= \min(f(x),g(y))$$

is replaced by:

$$t =l= f(x)$$
$$t =l= g(y)$$

and is effective as long as the objective maximizes $t$. If not, this smooth approximation for $\min(f(x),g(y))$ can be used:

$$\frac{[f(x) + g(y) - \sqrt{sqr(f(x)-g(y)) + sqr(delta))}]}{2}$$

### 4.18 Data Exchange with Other Applications

- Data Exchange with Text Files
- Data Exchange with Microsoft Excel
- Data Exchange with Databases
  - Data Exchange with DB2
  - Data Exchange with MS Access
  - Data Exchange with Oracle
  - Data Exchange with MySQL
  - Data Exchange with SQL Server
  - Data Exchange with SQLite
  - Data Exchange with Sybase
4.18.1 Data Exchange with Text Files

This tutorial describes some ways on how to exchange data between GAMS and text files (usually in ASCII format).

4.18.1.1 Reading Text Files During Compilation

GAMS can read arbitrary text files during compile time by inserting them into the compiler input stream. The file content is then assumed to be GAMS code. Thus, including a text file within a data statement (see also Data Entry: Parameters, Scalars and Tables) allows for an easy way to include data from a text file, as long as the syntax in the text file can be understood by the GAMS compiler. This way, model specification and data input can be separated into different files.

The $include compile-time command is used to instruct the GAMS compiler to include the context of a different file at the current position of the input stream. As a result, the GAMS code behaves as if the $include statement has been replaced by the content of the file to be included. This can be very handy when including data from a separate text file. For instance, when data for a table is actually coming from another environment, one could replace the TABLE statement by an include statement. A GAMS table is in fact very well suited for a human being to be read or written, but it is rather awkward for programs to generate (e.g., the numbers have to be approximately below the corresponding headers). Therefore, often parameters are used and long series of assignment statements are generated. For instance, consider the following fragment from model [TRNSPORT]:

Table d(i,j) 'distance in thousands of miles'
    new-york      chicago    topeka
seattle       2.5         1.7     1.8
san-diego     2.5         1.8     1.4 ;

When the data for this table is coming from a program it is more convenient to say in the main program:

Parameter d(i,j) 'distance in thousands of miles';
$include data.inc
display d;

and to have the include file data.inc contain the machine generated statements:

d("seattle","new-york") = 2.5;
d("san-diego","new-york") = 2.5;
d("seattle","chicago") = 1.7;
d("san-diego","chicago") = 1.8;
d("seattle","topeka") = 1.8;
d("san-diego","topeka") = 1.4;

In fact, GAMS can deal quite comfortably with a large number of such assignment statements.

Note, that since the included file is considered as part of the GAMS input stream, it is also echoed in the listing file. When including large text files with data statements, echoing these files in the listing file can be undesired. To suppress echoing to the listing file, the $include statement can be surrounded by $offlisting and $onlisting instructions:
parameter d(i,j) 'distance in thousands of miles';
$offlisting
$include data.inc
$onlisting
display d;

In the listing file, line numbers are skipped where the $offlisting is in effect.

In some cases it may be more convenient to use the initialization syntax of parameters. That is, the main GAMS file could contain the fragment:

parameter d(i,j) 'distance in thousands of miles' /
$include data2.inc
/;
display d;

and the data file contains the following records:

seattle .new-york 2.5
san-diego.new-york 2.5
seattle .chicago 1.7
san-diego.chicago 1.8
seattle .topeka 1.8
san-diego.topeka 1.4

This approach is preferable for large data sets as it is more efficient for GAMS.

Note

Tables and parameters are handled exactly the same way by GAMS internally. The only difference is in the specification of data.

When using a table statement, data can also be specified in CSV (Comma-separated values) format if the $ondelim command has been issued. This format can, for instance, be generated by spreadsheet programs.

As example, consider again the following fragment from model [TRNSPORT]:

Table d(i,j) 'distance in thousands of miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

A file data.csv that specifies the data of this table in CSV format would have the content

,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4

Notice the empty first element in the first line, which corresponds to the top-left blank in the above table.

This file can now be included directly into GAMS by using the $ondelim and $offdelim commands:

Table d(i,j) 'distance in thousands of miles'
$ondelim
$include data.csv
$offdelim
;
Attention

The $ondelim command only enables the use of commas as a separation symbol. The use of a different separation character is not supported by GAMS. This can become an issue if, for example, a CSV file has been generated with a language setting where commas are used as decimal point and semicolons are used for separating entries in the CSV file. In such a situation, data.csv would have the content

, new-york; chicago; topeka
seattle; 2,5; 1,7; 1,8
san-diego; 2,5; 1,8; 1,4

Therefore, such text files need to be preprocessed before read into GAMS. The following code uses the POSIX tool tr to (simultaneously) replace commas by dots and semicolons by commas:

$call "tr ,; ., < data.csv > data2.csv"
Table d(i,j) 'distance in thousands of miles';
$ondelim
$include data2.csv
$offdelim

CSV files can also be used to input higher dimensional data into GAMS. For instance, consider the 3 dimensional table yieldtl from model [TURKEY]:

Table yieldtl(l,cl,ty) 'livestock yield time series (kg per head)'

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sheep.meat</td>
<td>10.60</td>
<td>11.42</td>
<td>10.60</td>
<td>9.38</td>
<td>8.97</td>
<td>6.93</td>
</tr>
<tr>
<td>sheep.milk</td>
<td>23.7</td>
<td>24.1</td>
<td>24.2</td>
<td>24.0</td>
<td>23.9</td>
<td></td>
</tr>
<tr>
<td>sheep.wool</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>sheep.hide</td>
<td>0.5</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>goat.milk</td>
<td>37.7</td>
<td>38.1</td>
<td>38.2</td>
<td>38.3</td>
<td>37.8</td>
<td></td>
</tr>
<tr>
<td>goat.wool</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>goat.hide</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>angora.milk</td>
<td>1.77</td>
<td>1.77</td>
<td>2.66</td>
<td>2.21</td>
<td>1.77</td>
<td>1.77</td>
</tr>
<tr>
<td>angora.wool</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.4</td>
</tr>
<tr>
<td>angora.hide</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>cattle.milk</td>
<td>210.0</td>
<td>208.1</td>
<td>219.8</td>
<td>213.8</td>
<td>214.8</td>
<td>217.5</td>
</tr>
<tr>
<td>cattle.hide</td>
<td>3.3</td>
<td>3.4</td>
<td>2.9</td>
<td>3.0</td>
<td>2.6</td>
<td>3.3</td>
</tr>
<tr>
<td>buffalo.meat</td>
<td>43.73</td>
<td>45.42</td>
<td>40.61</td>
<td>37.21</td>
<td>32.20</td>
<td>32.68</td>
</tr>
<tr>
<td>buffalo.milk</td>
<td>267.1</td>
<td>269.2</td>
<td>263.8</td>
<td>219.6</td>
<td>275.5</td>
<td>285.1</td>
</tr>
<tr>
<td>buffalo.hide</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>poultry.meat</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
</tr>
<tr>
<td>poultry.egg</td>
<td>62.4</td>
<td>62.2</td>
<td>64.2</td>
<td>78.3</td>
<td>76.4</td>
<td>73.3</td>
</tr>
</tbody>
</table>

When the data for this parameter is prepared by another application (such as a relational database), it may be more convenient to write it out in a comma-separated value form, e.g., a file data.csv could have the content

"sheep","meat","1974",10.60
"sheep","meat","1975",11.42
"sheep","meat","1976",10.60
Including such formatted data into a GAMS model is possible by using $ondelim for Parameter Data for Higher Dimensions. Thus, the GAMS code would be

```gams
Parameter yieldtl(l,cl,ty) 'livestock yield time series (kg per head)'
/ $ondelim
$include data.csv
$offdelim
/;
```

### 4.18.1.2 Writing Text During Compilation

The commands $echo, $onecho, and $offecho send text to named files during compilation. $echo sends one line and is invoked using the syntax:

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'text to be sent' >> externalfile
```

The use of "">

```
$onecho > externalfile
line 1 of text to be sent
line 2 of text to be sent
...
lst line of text to be sent
$offecho
```

A typical example for the usage of these commands is the generation of a solver options file.

Additionally, the $log command can be used to send messages to the log file during compilation.
4.18.1.3 Writing Text Files During Execution

The put writing facility allows customized text output. This is a fairly complex but powerful and flexible report writing facility.

Assume that the following GAMS code is added to the end of model [TRNSPORT]. It instructs GAMS to write the model and the solve status together with levels of the decision variables to a file results.txt:

File results / results.txt /;
put results;
put "Model status", transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
    put i.tl, j.tl, x.l(i,j) /);
putclose;

First, a file object results is declared by using the File statement. The data of the file statement specifies the name of the file (results.txt). Next, line put results; instructs GAMS that for the following put statements, the file results should be used. In the following, the model status and the solve status are written, together with some descriptive text. These model attributes are set by a solve statement. The character '/' instructs GAMS to add a linebreak (newline character) to the results file. The next thing to write out are some variable values. Here, first the level value of variable z is written, followed by a loop that writes for each element of sets i and j the name of the elements (accessed via the .tl attribute) and the level value of x(i,j). Finally, the putclose; statement instructs GAMS to close the current file. This will ensure that possibly cached data is flushed to the file.

The output will look like:

Model status 1.00
Solver status 1.00
Objective 153.67
Shipments
    seattle  new-york  50.00
    seattle  chicago   300.00
    seattle  topeka    0.00
    san-diego new-york 275.00
    san-diego chicago  0.00
    san-diego topeka  275.00

This form can be hard to read by other applications, e.g., because some space characters are to be considered as separators, while others are really part of a string (e.g. "Model status"). However, by using the print control option .pc of the put writing facility, comma-separated value files can be written. That is, by adding the line results.pc = 5;; i.e.,

File results / results.txt /;
    results.pc = 5;
put results;
put "Model status", transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
    put i.tl, j.tl, x.l(i,j) /);
putclose;
4.18 Data Exchange with Other Applications

one obtains the following output:

"Model status",1.00
"Solver status",1.00
"Objective",153.67
"Shipments"
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00

If several parameters of the same dimension should be written to a file in a customized format, the put statements can become rather repetitive. For example, the level and marginal values of variable \(x\) and the parameters \(c\) and \(d\) from model [TRNSPORT] should be written. This can be coded easily as:

```plaintext
file results / results.txt /;
results.pc = 5;
put results;
loop((i,j), put "distance", i.tl, j.tl, d(i,j) / );
loop((i,j), put "cost", i.tl, j.tl, c(i,j) / );
loop((i,j), put "levels", i.tl, j.tl, x.l(i,j) / );
loop((i,j), put "marginals", i.tl, j.tl, x.m(i,j) /);
putclose;
```

A separation of the code that writes the data (loop and put statements) from the data that is written (descriptive text, set names, etc.) can be achieved by using $batinclude. This command works similar to the $include statement (see also Section Reading Text Files During Compilation above), but allows for additional arguments (separated by blanks). While including the file, markers %1, %2, etc., are replaced by the value of the 1st, 2nd, etc., argument.

By using $batinclude, the above example could be simplified to

```plaintext
file results / results.txt /;
results.pc = 5;
put results;
$batinclude put.inc distance i j d
$batinclude put.inc cost i j c
$batinclude put.inc level i j x.l
$batinclude put.inc marginal i j x.m
putclose;
```

where the file put.inc contain the actual loop and put statements:

```plaintext
loop((%2,%3), put "%1", %2.tl, %3.tl, %4(%2,%3) / );
```

Finally, the The Put_Utility Statement is referred, which is loosely connected to the put statement and allows the dynamic generation of file names, etc.
4.18.2 Data Exchange with Microsoft Excel

This tutorial gives an overview on how to exchange data between GAMS and Microsoft Excel.

GAMS can communicate with Microsoft Excel via GDX (GAMS Data Exchange) files. In order to write data from GAMS and to Excel, selected GAMS data can be written into a GDX file and then to an Excel file: GAMS -> GDX -> Excel. Similarly selected Excel data can be written to a GDX file and then read into GAMS: Excel -> GDX -> GAMS.

Some of GAMS/Excel data exchange tools that provide functionality to exchange data between GAMS and Excel are also discussed in the section Data Exchange Tools. The data exchange between GAMS and a CSV (Comma-separated values) file format and GAMS is covered in Data Exchange with Text Files.

4.18.2.1 From GAMS to Excel

Consider the following modification of the [TRNSPORT] model from the gams model library.

Sets
i 'canning plants' / seattle, san-diego /
j 'markets' / new-york, chicago, topeka / ;

Parameters
a(i) 'capacity of plant i in cases'
  / seattle 350
  san-diego 600 /
b(j) 'demand at market j in cases'
  / new-york 325
  chicago 300
  topeka 275 /

Table d(i,j) 'distance in thousands of miles'
  new-york 2.5 1.7 1.8
  san-diego 2.5 1.8 1.4 ;

Scalar f 'freight in dollars per case per thousand miles' /90/ ;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f * d(i,j) / 1000 ;

Variables
x(i,j) 'shipment quantities in cases'
z 'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
cost 'define objective function'
supply(i) 'observe supply limit at plant i'
demand(j) 'satisfy demand at market j'
cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;
4.18 Data Exchange with Other Applications

supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;
Model transport /all/ ;
Solve transport using lp minimizing z ;
Display x.l, x.m ;

*=== Export to Excel using GDX utilities
*=== First unload to GDX file (occurs during execution phase)
execute_unload "results.gdx" x.L x.M
*=== Now write to variable levels to Excel file from GDX
*=== Since we do not specify a sheet, data is placed in first sheet
execute 'gdxxrw.exe results.gdx o=results.xlsx var=x.L'
*=== Write marginals to a different sheet with a specific range
execute 'gdxxrw.exe results.gdx o=results.xlsx var=x.M rng=NewSheet!f1:i4'

After the solve statement, the data (x.L and x.M) from variable x can be written into a GDX file during the execution time using the command executeUnload:

execute_unload "results.gdx" x.L x.M

The executeunload command above is executed during the actual execution phase to create a GDX file called results.gdx. The solution x and the marginals of x in the GDX file can be written to the Excel file results.xlsx using GDXXRW tool:

execute 'gdxxrw.exe results.gdx var=x.L'
execute 'gdxxrw.exe results.gdx var=x.M rng=NewSheet!f1:i4'

For the first call for x.L, there is no range specified and the data is written in cell A1 and beyond in the first available sheet. For the second call for marginals x.M, data will be written to cells F1:I4 in the sheet named NewSheet.

Note that GAMS can also write data into a GDX file during compile time. It is also possible to convert data stored in a GDX file into an Excel file spreadsheets using GDX2XLS tool and to write GAMS data to standard output formatted as a GAMS program with data statements using GDXDUMP tool.

4.18.2.2 From Excel to GAMS

Consider the following modification of the [TRNSPORT] model from the gams model library and the file results.xlsx file created from the previous example.

Sets
i 'canning plants’ / seattle, san-diego /
j 'markets’ / new-york, chicago, topeka / ;

Parameters
a(i) 'capacity of plant i in cases'
   / seattle 350
   san-diego 600 /

b(j) 'demand at market j in cases'
   / new-york 325
   chicago 300
   topeka 275 / ;

Table d(i,j) 'distance in thousands of miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Scalar f 'freight in dollars per case per thousand miles' /90/ ;

Parameter c(i,j) 'transport cost in thousands of dollars per case' ;

\[ c(i,j) = \frac{f \times d(i,j)}{1000} ; \]

Variables
x(i,j) 'shipment quantities in cases'
z 'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
cost 'define objective function'
supply(i) 'observe supply limit at plant i'
demand(j) 'satisfy demand at market j' ;

\[ \text{cost} \quad z = \text{sum}(i,j, c(i,j) \times x(i,j)) ; \]

\[ \text{supply}(i) \quad \text{sum}(j, x(i,j)) = a(i) ; \]

\[ \text{demand}(j) \quad \text{sum}(i, x(i,j)) = b(j) ; \]

Model transport /all/ ;

==== Import from Excel using GDX utilities

==== First unload to GDX file (occurs during compilation phase)
$call gdxxrw.exe results.xlsx par=Level rng=sheet1!A1:D3

==== Now import data from GDX
Parameter Level(i,j);
$gdxin results.gdx
$load Level
$gdxin

==== Fix variables to values from Excel file
\[ x.FX(i,j) = Level(i,j) ; \]
display Level, x.L;

Solve transport using lp minimizing z ;
Display x.l, x.m;

The data in the Excel file can be loaded into a GDX file using the \$call command and GDXXRW tool:

\$call gdxrw.exe results.xlsx par=Level rng=A1:D3

The command \$call above executes a program called GDXXRW during compile time. The GDXXRW reads data from the range A1:D3 in results.xlsx into a GAMS parameter called Level in the GDX file results.gdx. As an output GDX file is not specified when calling GDXXRW, the output file will be derived from the input file by changing the file extension of the input file and removing any path information.

To import data from a GDX file into a parameter, the parameter must be defined over appropriate sets before read. The data from a GDX file can be read during the compile time using the commands \$gdxin and \$load:

Parameter Level(i,j);
\$gdxin results.gdx
\$load Level
\$gdxin

The first command \$gdxin specifies the name of the GDX file results.gdx to be read. The command \$load reads parameter Level from the GDX file. The second command \$gdxin closed the GDX file.

GAMS can read from a GDX file either during compile time or during execution time. See Example 4 - Reading a GDX File when reading data with domain information and Example 5 - Reading a GDX File when reading from a GDX file during execution time.

Note that it is also possible to write all worksheets of an Excel workbook into a GDX file using XLSDUMP tool.

4.18.2.3 Data Exchange Tools

There are a number of tools that provide functionality to exchange data between GAMS and an Excel file. This section discusses some of the data exchange tools with some examples. The complete list of the tools can be found at GAMS/Excel Data Exchange tools.

4.18.2.3.1 GDXXRW  GDXXRW is a tool to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a 'GDX' file, or read from a 'GDX' file, and write the data to different ranges in a spreadsheet.

How to use GDXXRW to exchange data between GAMS and Excel is covered in the section From GAMS to Excel and the section From Excel to GAMS. More details on usage and examples of GDXXRW tool is covered in GDXXRW.
XLS2GMS is a simple utility that allows you to extract data from an Excel spreadsheet and convert it into a GAMS include file. XLS2GMS can be run interactively or in batch mode.

Consider the Excel data from the following spreadsheet:

The data can be imported from the Excel file into a GAMS include file by calling XLS2GMS tool and inserted an include file as parameter data elements using the command $include:

```gams
set ssi /
   'new york', 'washington dc', 'los angeles', 'san francisco'
/;
parameter ssdata(ssi) /
$call =d:\util\xls2gms I="c:\my documents\test2.xlsx" B O=d:\tmp\x.inc
$include d:\tmp\x.inc
$display ssdata;
```

Notice the B parameter, which is needed as there are embedded blanks in the labels.

Sometimes a translation between the labels used in the model and the ones used in the is needed. One way to do this is to use a mapping set in GAMS. Suppose the rest of the model is defined in terms of the set I which is defined as:

```gams
set i / ny, dc, la, sf/;
```

To map a parameter data defined over this set, the following simple GAMS fragment can be used:
set map(i,ssi) mapping set / 
  ny.'new york'
  dc.'washington dc'
  la.'los angeles'
  sf.'san francisco'
/;
display map;

parameter data(i);
data(i) = sum(map(i,ssi), ssdata(ssi));
display data;

4.18.2.3 SQL2GMS In some cases it is convenient to consider tabular data in an Excel spreadsheet as a database table and to import it via GDX file using the SQL2GMS tool.

Consider the following spreadsheet:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>year</td>
<td>loc</td>
<td>prod</td>
<td>sales</td>
<td>profit</td>
</tr>
<tr>
<td>2</td>
<td>1997</td>
<td>la</td>
<td>hardware</td>
<td>80</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1997</td>
<td>la</td>
<td>software</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1997</td>
<td>nyc</td>
<td>hardware</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>1997</td>
<td>nyc</td>
<td>software</td>
<td>130</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>1997</td>
<td>sfö</td>
<td>hardware</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>1997</td>
<td>sfö</td>
<td>software</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>1997</td>
<td>was</td>
<td>hardware</td>
<td>80</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1997</td>
<td>was</td>
<td>software</td>
<td>90</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>1998</td>
<td>la</td>
<td>hardware</td>
<td>88</td>
<td>5,25</td>
</tr>
<tr>
<td>11</td>
<td>1998</td>
<td>la</td>
<td>software</td>
<td>66</td>
<td>10,5</td>
</tr>
<tr>
<td>12</td>
<td>1998</td>
<td>nyc</td>
<td>hardware</td>
<td>110</td>
<td>15,75</td>
</tr>
<tr>
<td>13</td>
<td>1998</td>
<td>nyc</td>
<td>software</td>
<td>143</td>
<td>26,25</td>
</tr>
<tr>
<td>14</td>
<td>1998</td>
<td>sfö</td>
<td>hardware</td>
<td>55</td>
<td>9,45</td>
</tr>
<tr>
<td>15</td>
<td>1998</td>
<td>sfö</td>
<td>software</td>
<td>66</td>
<td>6,3</td>
</tr>
<tr>
<td>16</td>
<td>1998</td>
<td>was</td>
<td>hardware</td>
<td>88</td>
<td>7,35</td>
</tr>
<tr>
<td>17</td>
<td>1998</td>
<td>was</td>
<td>software</td>
<td>99</td>
<td>8,4</td>
</tr>
</tbody>
</table>

This table can be read using an SQL query:

```sql
SELECT year,loc,prod,'sales',sales FROM [profitdata$] 
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
```

The table name is equal to the sheet name(profitdata). We can pass the query to the Excel ODBC driver using the tool SQL2GMS tool as follows:
set y 'years' /1997,1998/;
set c 'city' /la,nyc,sfo,was/;
set p 'product' /hardware,software/;
set k 'key' /sales,profit/;

$onecho > excelcmd.txt
c=DRIVER=Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb);dbq=%system.fp%profit.xlsx;
q=SELECT year,loc,prod,'sales',sales FROM [profitdata$] \ 
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
x=fromexcel.gdx
$offecho
$call =sql2gms @excelcmd.txt
parameter d(y,c,p,k) ;
$gdxin excel.gdx
$load d=p
display d;

and the DISPLAY results will be:

--- 21 PARAMETER d FROM SQL2GMS

INDEX 1 = 1997

<table>
<thead>
<tr>
<th></th>
<th>sales</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>la</td>
<td>hardware 80.000</td>
<td>5.000</td>
</tr>
<tr>
<td>la</td>
<td>software 60.000</td>
<td>10.000</td>
</tr>
<tr>
<td>nyc</td>
<td>hardware 100.000</td>
<td>15.000</td>
</tr>
<tr>
<td>nyc</td>
<td>software 130.000</td>
<td>25.000</td>
</tr>
<tr>
<td>sfo</td>
<td>hardware 50.000</td>
<td>9.000</td>
</tr>
<tr>
<td>sfo</td>
<td>software 60.000</td>
<td>6.000</td>
</tr>
<tr>
<td>was</td>
<td>hardware 80.000</td>
<td>7.000</td>
</tr>
<tr>
<td>was</td>
<td>software 90.000</td>
<td>8.000</td>
</tr>
</tbody>
</table>

INDEX 1 = 1998

<table>
<thead>
<tr>
<th></th>
<th>sales</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>la</td>
<td>hardware 88.000</td>
<td>5.250</td>
</tr>
<tr>
<td>la</td>
<td>software 66.000</td>
<td>10.500</td>
</tr>
<tr>
<td>nyc</td>
<td>hardware 110.000</td>
<td>15.750</td>
</tr>
<tr>
<td>nyc</td>
<td>software 143.000</td>
<td>26.250</td>
</tr>
<tr>
<td>sfo</td>
<td>hardware 55.000</td>
<td>9.450</td>
</tr>
<tr>
<td>sfo</td>
<td>software 66.000</td>
<td>6.300</td>
</tr>
<tr>
<td>was</td>
<td>hardware 88.000</td>
<td>7.350</td>
</tr>
<tr>
<td>was</td>
<td>software 99.000</td>
<td>8.400</td>
</tr>
</tbody>
</table>

4.18.2.3.4 **GDXVIEWER**  
GDXVIEWER is a tool to view and convert data contained in GDX files. It can also export to csv, xlsx, xml-files and pivot tables. The usage and examples are covered in GDXVIEWER.

4.18.2.3.5 **GDX2XLS**  
GDX2XLS tool to convert the contents of a GDX file into an Excel file or an xml-file. The usage and examples are covered in GDX2XLS.
4.18.3  Data Exchange with Databases

This tutorial provides a guidance on how to exchange data between GAMS and various Database Management System.

4.18.3.1  Data Exchange with DB2

DB2 is one of IBM's relational database management systems.

4.18.3.1.1  Import from DB2  

DB2 has an EXPORT command that can be used to generate comma delimited files. An example of a DB2 session illustrating this is shown below:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Type name</th>
<th>Length</th>
<th>Scale</th>
<th>Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCA</td>
<td>SYIBM</td>
<td>VARCHAR</td>
<td>10</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>LOCB</td>
<td>SYIBM</td>
<td>VARCHAR</td>
<td>10</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>DISTANCE</td>
<td>SYIBM</td>
<td>DOUBLE</td>
<td>8</td>
<td>0</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3 record(s) selected.

<table>
<thead>
<tr>
<th>LOCA</th>
<th>LOCB</th>
<th>DISTANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>new-york</td>
<td>+2.500000000000000E+000</td>
</tr>
<tr>
<td>seattle</td>
<td>chicago</td>
<td>+1.700000000000000E+000</td>
</tr>
<tr>
<td>seattle</td>
<td>topeka</td>
<td>+1.800000000000000E+000</td>
</tr>
<tr>
<td>san-diego</td>
<td>new-york</td>
<td>+2.500000000000000E+000</td>
</tr>
<tr>
<td>san-diego</td>
<td>chicago</td>
<td>+1.800000000000000E+000</td>
</tr>
<tr>
<td>san-diego</td>
<td>topeka</td>
<td>+1.400000000000000E+000</td>
</tr>
</tbody>
</table>

6 record(s) selected.

Number of rows exported: 6
The resulting data file `export.txt` will look like:

```
"seattle","new-york",+2.50000000000000E+000
"seattle","chicago",+1.70000000000000E+000
"seattle","topeka",+1.80000000000000E+000
"san-diego","new-york",+2.50000000000000E+000
"san-diego","chicago",+1.80000000000000E+000
"san-diego","topeka",+1.40000000000000E+000
```

This file can be read into GAMS using `$include`:

```gams
parameter d(i,j) 'distance in thousands of miles' /
   $ondelim
   $include export.txt
   $offdelim
;/
display d;
```

### 4.18.3.1.2 Export to DB2

DB2 has an `IMPORT` command that can read delimited files. As an example consider the file generated by GAMS `PUT` statements:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

A transcript of a DB2 session to read this file, is given below:

```
------------------------------------- Command Entered -------------------------------------
create table results(loca varchar(10) not null,
                       locb varchar(10) not null,
                       shipment double not null) ;
-------------------------------------------------------------------------------------------
DB20000I The SQL command completed successfully.

------------------------------------- Command Entered -------------------------------------
import from c:\tmp\import.txt of del insert into results ;
-------------------------------------------------------------------------------------------
SQL3109N The utility is beginning to load data from file "c:\tmp\import.txt".
SQL3110N The utility has completed processing. "6" rows were read from the input file.
SQL3221W ...Begin COMMIT WORK. Input Record Count = "6".
SQL3222W ...COMMIT of any database changes was successful.
SQL3149N "6" rows were processed from the input file. "6" rows were successfully inserted into the table. "0" rows were rejected.
```
4.18 Data Exchange with Other Applications

Number of rows read = 6
Number of rows skipped = 0
Number of rows inserted = 6
Number of rows updated = 0
Number of rows rejected = 0
Number of rows committed = 6

For very large data sets it is advised to use the LOAD command:

------------------------------------- Command Entered -------------------------------------
load from c:\tmp\import.txt of del insert into results ;
-------------------------------------------------------------------------------------------

SQL3501W  The table space(s) in which the table resides will not be placed in 
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file "c:\tmp\import.txt".

SQL3500W  The utility is beginning the "LOAD" phase at time "03-20-2000
18:11:50.213782".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing. "6" rows were read from the 
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "6".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "03-20-2000
18:11:50.337092".

Number of rows read = 6
Number of rows skipped = 0
Number of rows loaded = 6
Number of rows rejected = 0
Number of rows deleted = 0
Number of rows committed = 6

For smaller data sets one can also generate a series of INSERT statements using the PUT facility.

4.18.3.2 Data Exchange with MS Access

Microsoft Office Access, previously known as Microsoft Access, is a relational database management 
system from Microsoft. It is a member of the Microsoft Office system.

4.18.3.2.1 Import from MS Access

MDB2GMS  MDB2GMS is a tool to convert data from an Microsoft Access database 
into GAMS readable format. The source is an MS Access database file (*.mdb or *.accdb) and the target 
is a GAMS Include File or a GAMS GDX File. MDB2GMS is part of the GAMS Data eXchange Tools, 
see documentation for more information.
4.18.3.2.1.2 SQL2GMS  SQL2GMS is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File or a GAMS GDX File. SQL2GMS is part of the GAMS Data eXchange Tools, see documentation for more information.

4.18.3.2.1.3 CSV Files  Microsoft Access can export tables into comma delimited text files using its Save As/Export menu. Suppose we have the following table:

<table>
<thead>
<tr>
<th>loca</th>
<th>locb</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>new-york</td>
<td>2.5</td>
</tr>
<tr>
<td>seattle</td>
<td>chicago</td>
<td>1.7</td>
</tr>
<tr>
<td>seattle</td>
<td>topeka</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>new-york</td>
<td>2.5</td>
</tr>
<tr>
<td>san-diego</td>
<td>chicago</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>topeka</td>
<td>1.4</td>
</tr>
</tbody>
</table>

After choosing Save As/Export and selecting Text Files we get the following window:
Just using the default settings, we get the following file:

"seattle","new-york",2.50
"seattle","chicago",1.70
"seattle","topeka",1.80
"san-diego","new-york",2.50
"san-diego","chicago",1.80
"san-diego","topeka",1.40

which can be handled in GAMS by $ondelim/$offdelim and $include:

```gams
parameter d(i,j) 'distance in thousands of miles' / $ondelim
   $include dist.txt
   $offdelim /;
display d;
```

### 4.18.3.2.1.4 Import Dates from Access

GAMS dates are one day off when importing from MS Access. Suppose we have an MS Access table with one single date column:

```plaintext
datefield
--------
3/12/2007
3/13/2007 10:00:00 AM
3/14/2007 8:30:00 PM
```
The date data above can be imported into GAMS using $call and MDB2GMS tool as follows:

```
$call =mdb2gms I="%system.fp%Sample.accdb" Q="select datefield,Cdbl(datefield) from datetable" O=x.inc

parameter p(*) /
  $include x.inc
/
display p;
alias(*,i);
parameter q(*,+);
loop(i$p(i),
  q(i,'year') = gyear(p(i));
  q(i,'month') = gmonth(p(i));
  q(i,'day') = gday(p(i));
  q(i,'hour') = ghour(p(i));
  q(i,'minute') = gminute(p(i));
);
display q;
```

Note that the Cdbl() function converts the date to a floating point number (double precision). The generated include file looks like:

```
* -----------------------------------------------------
* MDB2GMS Version 2.8, January 2007
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 3.6
* Jet version: 4.0
* Database: D:\mdb2gms\examples\Sample.accdb
* Query: select datefield,Cdbl(datefield) from datetable
* -----------------------------------------------------
'
3/12/2007 39153
3/13/2007 10:00:00 AM 39154.4166666667
3/14/2007 8:30:00 PM 39155.8541666667
* -----------------------------------------------------
```

which looks o.k. However, when we look at the GAMS results in the listing file we see:

```
---- 28 PARAMETER p
3/12/2007 39153.000, 3/13/2007 10:00:00 AM 39154.417, 3/14/2007 8:30:00 PM 39155.854
```

```
---- 39 PARAMETER q

          | year | month | day  | hour | minute |
----------|------|-------|------|------|--------|
3/12/2007 | 2007 | 3.000 | 13.000 |
3/13/2007 | 2007 | 3.000 | 14.000 | 10.000 |
3/14/2007 | 2007 | 3.000 | 15.000 | 20.000 | 30.000 |
```

Clearly the dates are off by one day: see the column day. We can fix this problem in different places, e.g. in the query or in the GAMS model by subtracting 1.0 from an imported date. This problem occurs not only in MS Access but also with other software packages.

4.18.3.2.2 Export to MS Access
4.18 Data Exchange with Other Applications

4.18.3.2.2.1 GDX2ACCESS  
GDX2ACCESS is a tool to dump the whole contents of a GDX file to a new MS Access file (.mdb or .accdb file). GDX2ACCESS is part of the GAMS Data eXchange Tools, see documentation for more information.

4.18.3.2.2.2 GDXVIEWER  
Access tables in MS Access files can be directly generated by the GDXVIEWER tool. The GDXVIEWER tool uses OLE automation to export data to an MS Access database. This means that MS Access needs to be installed for the Access Export facility to work. GDXVIEWER is part of the GAMS Data eXchange Tools, see documentation for more information.

4.18.3.2.2.3 VBScript  
VBScript is a scripting tool that can be used to talk to COM objects. In this case we use it to tell Access to import a CSV file.

```
$ontext
Import a table into MS Access using VBscript
$offtext
$if exist new.accdb $call del new.accdb
set i /i1*i10/;
alias (i,j);
parameter p(i,j);
p(i,j) = uniform(-100,100);
display p;
file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
   put i.tl, j.tl, p(i,j):12:8/
);
putclose;
execute "=cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
Wscript.Echo "Running script: access.vbs"
DbLangGeneral = ";:LANGID=0x0409;CP=1252;COUNTRY=0"
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=%system.fp%;FMT=Delimited].[data#csv]"
Wscript.Echo "Query : " & strSQL
Set oJet = CreateObject("DAO.DBEngine.36")
Wscript.Echo "Jet version : " & oJet.version
Set oDB = oJet.createDatabase("new.accdb",DbLangGeneral)
Wscript.Echo "Created : " & oDB.name
oDB.Execute strSQL
Set TableDef = oDB.TableDefs("mytable")
Wscript.Echo "Rows inserted in mytable : " & TableDef.RecordCount
oDB.Close
Wscript.Echo "Done"
$offecho
```

The CSV file contains a header row with the names of the fields:

```
"i","j","p"
"i1","i1",-65.65057360
"i1","i2",68.65334160
"i1","i3",10.07507120
"i1","i4",-39.77241920
"i1","i5",-41.55757660
....
```
The text driver specification `HDR=Yes` makes sure the first row in the CSV file is treated specially. The log will look like:

```
U:\temp>gams vbaccess.gms
--- Job vbaccess.gms Start 01/28/08 16:57:37
GAMS Rev 149 Copyright (C) 1987-2007 GAMS Development. All rights reserved
...
--- Starting compilation
--- vbaccess.gms(4) 2 Mb
--- call del new.accdb
--- vbaccess.gms(38) 3 Mb
--- Starting execution: elapsed 0:00:00.109
--- vbaccess.gms(18) 4 Mb
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
...
Running script: access.vbs
Query : SELECT * INTO mytable FROM [Text;HDR=Yes;Database=U:\temp\;FMT=Delimited ].[data#csv]
Jet version : 3.6
Created : U:\temp\new.accdb
Rows inserted in mytable : 100
Done
--- Putfile f U:\temp\data.csv
*** Status: Normal completion
--- Job vbaccess.gms Stop 01/28/08 16:57:38 elapsed 0:00:00.609
U:\temp>
```

Please note that although the `$onecho/$offecho` is at the bottom of the GAMS file, the file `access.vbs` is created at compile time. I.e. before the executable statements like `PUT`, `EXECUTE` are executed.

### 4.18.3.2.2.4 JScript

The same script using `JScript` is similar to the one with VScript. We only price the script itself.

```jscript
$ontext
Import a table into MS Access using JScript
$offtext
$if exist new.accdb $call del new.accdb

set i /i1*i10/;
alias (i,j);
parameter p(i,j);

p(i,j) = uniform(-100,100);
display p;

file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
    put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.js";
```
4.18 Data Exchange with Other Applications

```javascript
$onecho > access.js
// this is a JScript script
WScript.Echo("Running script: access.js");

dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0";
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=.;FMT=Delimited].[data#csv]";
WScript.Echo("Query : ", strSQL);

oJet = new ActiveXObject("DAO.DBEngine.36");
WScript.Echo("Jet version : ", oJet.version);

oDB = oJet.createDatabase("new.accdb", dbLangGeneral);
WScript.Echo("Created : ", oDB.name);

oDB.Execute(strSQL);
TableDef = oDB.TableDefs("mytable");
WScript.Echo("Rows inserted in mytable : ", TableDef.RecordCount);

oDB.Close();
WScript.Echo("Done");
$offecho
```

4.18.3.2.2.5 Combining GDX2ACCESS and VBscript

Data in a GDX file do not contain domain information. I.e. a parameter c(i,j) is really stored as c(*,*)
As a result GDX2ACCESS will invent field names like dim1, dim2, Value. In some cases this may not be convenient, e.g. when more descriptive field names are required. We will show how a small script in VBscript can handle this task. The script will rename the fields dim1, dim2, Value in table c to i, j, and transportcost.

```vbscript
$call "gamslib 1"
$include transport.gms
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
execute_unload "c.gdx",c;
* move to access database
* column names are dim1,dim2
* execute "+gdx2access c.gdx"
* rename columns
* execute "+cscript access.vbs"

$onecho > access.vbs
' this is a VBscript script
WScript.Echo "Running script: access.vbs"

' Office 2000 DAO version
' Change to local situation.
Set oDAO = CreateObject("DAO.DBEngine.36")
script.Echo "DAO version : " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%\c.accdb")
WScript.Echo "Opened : " & oDB.name
```

The above VBscript fragment needs to be adapted according to the DAO version available on the client machine. This can be implemented in a more robust fashion by letting MS Access find the DAO engine:

```
' this is a VBscript script
WScript.Echo "Running script: access.vbs"
set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version
Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name
Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name
' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"
oDB.Close
Wscript.Echo "Done"
```

Please note that the macro `%system.fp%` is replaced by GAMS by the working directory (this is the project directory when running GAMS from the IDE).

### 4.18.3.3 Data Exchange with MySQL

MySQL is a multi-threaded, multi-user SQL database management system.

#### 4.18.3.3.1 Import from MySQL

MySQL can write the results of a `SELECT` statement to a file as follows:
mysql> select * from dist;
+-------------------+----------+----------+
| loca              | locb     | distance |
+-------------------+----------+----------+
| seattle           | new-york | 50       |
| seattle           | chicago  | 300      |
| seattle           | topeka   | 0        |
| san-diego         | new-york | 275      |
| san-diego         | chicago  | 0        |
| san-diego         | topeka   | 275      |
+-------------------+----------+----------+
6 rows in set (0.01 sec)

mysql> select * from dist into outfile '/tmp/data.csv'
   -> fields terminated by ','
   -> optionally enclosed by '"'
   -> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)

The resulting CSV file looks like:

"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275

which can be read by GAMS directly. This approach can be automated as follows:

[erwin@localhost erwin]$ cat myscript
use test
select * from dist into outfile '/tmp/data.csv'
   fields terminated by ','
   optionally enclosed by '"'
   lines terminated by '\n';
[erwin@localhost erwin]$ cat x.gms
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;
$call 'mysql -u root < myscript'
parameter dist(i,j) /
$ondelim
$include /tmp/data.csv
$offdelim
/
display dist;
[erwin@localhost erwin]$ gams x
GAMS Rev 132 Copyright (C) 1987-2002 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC  G871201:0000XX-XXX
Free Demo, 202-342-0180, sales@gams.com, www.gams.com  DC9999
--- Starting compilation
--- x.gms(5) 1 Mb
--- call mysql -u root < myscript
--- .data.csv(6) 1 Mb
--- x.gms(15) 1 Mb
The listing file shows that the table is read correctly:

```gams
set i /seattle, san-diego/
set j /new-york, chicago, topeka/;
parameter dist(i,j) /
   INCLUDE /tmp/data.csv
   "seattle","new-york",50
   "seattle","chicago",300
   "seattle","topeka",0
   "san-diego","new-york",275
   "san-diego","chicago",0
   "san-diego","topeka",275
   /;
display dist;
```

SEQ GLOBAL TYPE PARENT LOCAL FILENAME
1 1 INPUT 0 0 /home/erwin/x.gms
2 5 CALL 1 5 mysql -u root < myscript
3 8 INCLUDE 1 8 ./tmp/data.csv

**** 18 PARAMETER dist

<table>
<thead>
<tr>
<th>SEQ</th>
<th>GLOBAL</th>
<th>TYPE</th>
<th>PARENT</th>
<th>LOCAL</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>INPUT</td>
<td>0</td>
<td>0</td>
<td>/home/erwin/x.gms</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>CALL</td>
<td>1</td>
<td>5</td>
<td>mysql -u root &lt; myscript</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>INCLUDE</td>
<td>1</td>
<td>8</td>
<td>./tmp/data.csv</td>
</tr>
</tbody>
</table>

Instead of maintaining the MySQL script in a separate file, it can also be written by GAMS using `$onecho/$offecho` and a statement like:

```gams
$onecho > myscript
use test
select * from dist into outfile '/tmp/data.csv'
   fields terminated by ','
   optionally enclosed by "",
   lines terminated by '
';
$offecho
```

This will write the script at compile time.
4.18.3.3.2 Export to MySQL  GAMS can export data to MySQL by creating a script containing a series of SQL INSERT statements, as shown in section Oracle CSV Import.

It is noted that MySQL does have a REPLACE statement which is a useful blend of an INSERT and UPDATE statement: update a row if it already exists, otherwise insert it. This is not standard SQL however, so it can cause problems when moving to another database.

For larger result sets it may be better to use the LOAD DATA INFILE command. This command can read directly ASCII text files such as comma delimited CSV files.

Consider again the data file created by the PUT statement:

"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00

The following transcript shows how to import this into MySQL:

mysql> create table dist(loca varchar(10), locb varchar(10), distance double precision);
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| dist |
+----------------+
1 row in set (0.00 sec)

mysql> describe dist;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| loca     | varchar(10) | YES  |     | NULL    |       |
| locb     | varchar(10) | YES  |     | NULL    |       |
| distance | double      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> load data infile '/tmp/data.txt' into table dist
-> fields terminated by ',',
-> optionally enclosed by '"',
-> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
Records: 6 Deleted: 0 Skipped: 0 Warnings: 0

mysql> select * from dist;
+-----------+----------+----------+
| loca      | locb     | distance |
+-----------+----------+----------+
| seattle   | new-york | 50       |
| seattle   | chicago  | 300      |
| seattle   | topeka   | 0        |
| san-diego | new-york | 275      |
Note that we used no keys in our table definition. In practice it is of course highly recommended to define proper keys.

4.18.3.4 Data Exchange with Oracle

The Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a relational database management system (RDBMS) software product released by Oracle Corporation.

4.18.3.4.1 Import from Oracle

4.18.3.4.1.1 SQL*Plus To export an Oracle table a simple solution is to write an SQL*Plus script. E.g. if our table looks like:

```sql
SQL> describe dist;
Name Null? Type
----------------------------------- -------- ------------------------
LOCA NOT NULL VARCHAR2(10)
LOCB NOT NULL VARCHAR2(10)
DISTANCE NUMBER

SQL> select * from dist;
LOCA LOCB DISTANCE
---------- ---------- ---------
seattle new-york 2.5
topeka new-york 2.5
seattle chicago 1.7
topeka chicago 1.8
seattle topeka 1.4
san-diego new-york 2.5
san-diego chicago 1.8
san-diego topeka 1.4
6 rows selected.
```

then the following script will export this table:

```sql
set pagesize 0
set pause off
set heading off
spool data
select loca||','||locb||','||distance from dist;
spool off
```

The resulting data file "data.lst" will look like:
4.18 Data Exchange with Other Applications

seattle,new-york,2.5
seattle,chicago,1.7
seattle,topeka,1.8
san-diego,new-york,2.5
san-diego,chicago,1.8
san-diego,topeka,1.4

This almost looks like our data initialization syntax for parameters:

SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4

The only differences are in the delimiters that are being used. These differences are easily digested by GAMS once it is in ondelim mode. I.e. the following syntax can be used to read the data.lst file:

```gams
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include data.lst
$offdelim
/;
```

display d;

4.18.3.4.1.2 SQL2GMS An alternative way to import data from Oracle is to use the tool SQL2GMS which can talk to any database with an ADO or ODBC interface.

4.18.3.4.1.3 Import dates from Oracle databases and converting them to GAMS dates For most softwares it is easy to generate dates that GAMS can import and understand. The most common issue is that GAMS is one day off compared to Excel, Delphi, Access, ODBC etc. Oracle is somewhat more involved. First it is useful to have the date/time exported as a Julian date. This can be done with the following stored procedure:

```sql
-- julian representation of a date/time
-- Erwin Kalvelagen, feb 2007
create or replace function to_julian(d IN TIMESTAMP)
return number
is
begin
return to_number(to_char(d,'J')) + to_number(to_char(d,'SSSSS'))/86400;
end;
```

This function can be used to export dates as simple floating point numbers. In GAMS we need just a simple adjustment by adding a constant 'datediff' defined by:

```gams
scalar
    refdategams "march 16, 2006, 00:00"
    refdateoracle "march 16, 2006, 00:00" /2453811/
    datediff "difference between GAMS and Oracle date"
;
refdategams = jdate(2006,3,16);
datediff = refdategams-refdateoracle;
```

This trick has been applied in a complex scheduling application where dates are important data types that must be exchanged between the application logic and database tier and the optimization engine.
4.18.3.4.2 Export to Oracle

4.18.3.4.2.1 Oracle CSV Import  A familiar way of moving data into Oracle is to generate standard SQL INSERT statements. The PUT facility is flexible enough to handle this. For instance the following code:

```gams
file results /results.sql/;
results lw=0;
results nw=0;
put results;
loop((i,j),
    put "insert into result (loca, locb, shipment) ";
    put "values ("',i.tl,"','",j.tl,"','",x.l(i,j)),";"
);
putclose;
```

will generate these SQL statements:

```
insert into result (loca, locb, shipment) values ('seattle','new-york',50.00);
insert into result (loca, locb, shipment) values ('seattle','chicago',300.00);
insert into result (loca, locb, shipment) values ('seattle','topeka',0.00);
insert into result (loca, locb, shipment) values ('san-diego','new-york',275.00);
insert into result (loca, locb, shipment) values ('san-diego','chicago',0.00);
insert into result (loca, locb, shipment) values ('san-diego','topeka',275.00);
```

The .lw and .nw attributes for the put file indicate that no extra spaces around the labels and the numeric values are needed. These field width attributes have a default value of 12 which would cause the values to look like:

```
'seattle', 'new-york', 50.00
```

If the amount of data is large the utility SQL*Loader can be used to import comma delimited input. I.e. the GAMS code:

```gams
file results /results.txt/;
results pc=5;
put results;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

produces a file results.txt:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following SQL*Loader control file will read this file:

```
LOAD DATA
INFILE results.txt
INTO TABLE result
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(loca,locb,shipment)
```
4.18 Data Exchange with Other Applications

4.18.3.4.2.2 GDX to Oracle Database tables in an SQL RDBMS can be directly generated by the GDXVIEWER tool. The GDXVIEWER can use three methods to export to Oracle and other RDBMSs:

1. The direct ADO/ODBC link can create a new table and populate it.
2. The SQL INSERT script generator can create a script with a number of INSERT statements.
3. The SQL UPDATE script generator can create a script with a number of UPDATE statements.

4.18.3.5 Data Exchange with SQL Server

4.18.3.5.1 Import from SQL Server Microsoft SQL Server is Microsoft's flagship database. It comes in different flavors, including SQL Server, MSDE and SQL Server Express.

4.18.3.5.1.1 Using SQL2GMS A good way to import SQL server data into GAMS is using the SQL2GMS tool. Below is an example of its use:

```gams
$set commandfile commands.txt
$onecho > %commandfile%
C=provider=sqlolodb;data source=athlon\SQLExpress;Initial catalog=test;user id=sa;password=password
O=C:\WINNT\gamsdir\xx.inc
Q=SELECT * FROM x
$offecho
$call =sql2gms %commandfile%
parameter p(i,j) /
$include "C:\WINNT\gamsdir\xx.inc"
/;
display p;
```

4.18.3.5.1.2 Using the BCP utility and CSV files To export SQL Server data to CSV files we can use the BCP utility.

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>bcp test..results out x.csv -S athlon\sqlexpress -c -U sa -P password -t
Starting copy...
6 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total : 10 Average : (600.00 rows per sec.)
```

C:\Program Files\Microsoft SQL Server\90\Tools\binn>type x.csv
```
seattle,new-york,50.0
seattle,chicago,300.0
seattle,topeka,0.0
san-diego,new-york,275.0
san-diego,chicago,0.0
san-diego,topeka,275.0
```

It is somewhat more difficult to create a proper CSV file. A format specification file can help here. For an example see the next section on Data Exchange with Sybase. Other tools to export files include DTS (Data Transformation Services) and linked ODBC data sources.
Finally we can program directly an interface between SQL server tables and GAMS GDX files. A small example in C# can look like:

```csharp
# Example C# code for interfacing SQL server tables with GAMS GDX files

```
4.18 Data Exchange with Other Applications

C:\Program Files\Microsoft SQL Server\90\Tools\binn>bcp test..x in c:\winnt\gamsdir\results.csv \
-S athlon\sqlexpress -c -U sa -P password -t,

Starting copy...

6 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total : 10 Average : (600.00 rows per sec.)

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> select * from x;
3> go

<table>
<thead>
<tr>
<th>loca</th>
<th>locb</th>
<th>shipment</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>new-york</td>
<td>50</td>
</tr>
<tr>
<td>seattle</td>
<td>chicago</td>
<td>300</td>
</tr>
<tr>
<td>seattle</td>
<td>topeka</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>new-york</td>
<td>275</td>
</tr>
<tr>
<td>san-diego</td>
<td>chicago</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>topeka</td>
<td>275</td>
</tr>
</tbody>
</table>

(6 rows affected)
1> quit

Unfortunately, dealing with quoted strings is not straightforward with this tool (an example using a format file is shown in the next section on Data Exchange with Sybase). The same thing holds for BULK INSERT, which can read:

C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
seattle,new-york,50.00
seattle,chicago,300.00
seattle,topeka,0.00
san-diego,new-york,275.00
san-diego,chicago,0.00
san-diego,topeka,275.00

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> create table x(loca varchar(10), locb varchar(10), shipment float)
3> go

<table>
<thead>
<tr>
<th>loca</th>
<th>locb</th>
<th>shipment</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>new-york</td>
<td>50</td>
</tr>
<tr>
<td>seattle</td>
<td>chicago</td>
<td>300</td>
</tr>
<tr>
<td>seattle</td>
<td>topeka</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>new-york</td>
<td>275</td>
</tr>
<tr>
<td>san-diego</td>
<td>chicago</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>topeka</td>
<td>275</td>
</tr>
</tbody>
</table>

(6 rows affected)
1> select * from x
2> go
4.18.3.5.2.2 Export using the ODBC Text Driver  A slower but flexible way to load CSV files is to use a linked server through the ODBC Text Driver. First create an ODBC DSN using the Text Driver. This can be done through the ODBC Data Source Administrator Data Sources (ODBC) Then we can use the system procedure SP_AddLinkedServer.

---

C:\Program Files\Microsoft SQL Server\90\Tools\binn\ type c:\winnt\gamsdir\results.csv
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00

C:\Program Files\Microsoft SQL Server\90\Tools\binn\ type transport.sql
--
-- test database
--
use test

-- create table in SQL server
--
cREATE TABLE results (loca VARCHAR(10), locb VARCHAR(10), ship FLOAT)
GO

-- Create a linked server
--
EXEC sp_addlinkedserver txtsrv,'Jet 4.0','Microsoft.Jet.OLEDB.4.0','c:\winnt\gamsdir',NULL,'Text'
GO

-- copy data from text file c:\winnt\gamsdir\results.csv
--
INSERT INTO results(loca,locb,ship)
SELECT * FROM txtsrv...results#csv
GO

-- check if all arrived
--
SELECT * FROM results
GO

-- release linked server
--
EXEC sp_dropserver txtsrv
GO

--
-- clean up
--
drop table results

go

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\sqlexpress
1> :r transport.sql
Changed database context to 'test'.

(6 rows affected)
loca locb ship
---------- ---------- ------------------------
seattle new-york 50
seattle chicago 300
seattle topeka 0
san-diego new-york 275
san-diego chicago 0
san-diego topeka 275

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>

A slightly different approach is the following:

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\sqlexpress
1> create table t(loca varchar(10), locb varchar(10), ship float)
2> go
1> insert into t
2> select * from
3> OpenRowSet('Microsoft.Jet.OLEDB.4.0',
4> 'Text;Database=c:\winnt\gamsdir\;HDR=NO',
5> 'select * from results.csv')
6> go

(6 rows affected)
1> select * from t
2> go
loca locb ship
---------- ---------- ------------------------
seattle new-york 50
seattle chicago 300
seattle topeka 0
san-diego new-york 275
san-diego chicago 0
san-diego topeka 275

(6 rows affected)
1> drop table t
2> go
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>

This can be automated from GAMS as follows:
file results /results.csv/;
results.pc=5;
put results;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;

file sqlinsert /insert.sql/;
put sqlinsert;
put "use test"/;
put "insert into t select * from OpenRowSet('Microsoft.Jet.OLEDB.4.0',"
      "'Text;Database=c:\winnt\gamsdir\;HDR=NO','select * from results.csv')"/;
putclose;
execute '="C:\Progra~1\Microsoft SQL Server\90\Tools\binn\sqlcmd" -S athlon\SQLExpress -i insert.sql';

4.18.3.5.2.3 Export using the GDXVIEWER When using GDXVIEWER to export data to MS SQL server it is noted that MSSQL Server does not accept the default SQL type double for double precision numbers. You will need to set this setting to float or double precision.

When we export variable x from the transport model, we see:

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> go
Changed database context to 'test'.
1> select * from x
2> go
<table>
<thead>
<tr>
<th>dim1</th>
<th>dim2</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>new-york</td>
<td>50</td>
</tr>
<tr>
<td>seattle</td>
<td>chicago</td>
<td>300</td>
</tr>
<tr>
<td>seattle</td>
<td>topeka</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>new-york</td>
<td>275</td>
</tr>
<tr>
<td>san-diego</td>
<td>chicago</td>
<td>0</td>
</tr>
<tr>
<td>san-diego</td>
<td>topeka</td>
<td>275</td>
</tr>
</tbody>
</table>
(6 rows affected)
1> quit
C:\Program Files\Microsoft SQL Server\90\Tools\binn>

4.18.3.6 Data Exchange with SQLite

See GDX2SQLITE for more information.

4.18.3.7 Data Exchange with Sybase

4.18.3.7.1 Import from Sybase
4.18.3.7.1 Import using the bcp utility

Sybase is largely the same as SQL Server. For exporting ASCII files from a Sybase table, the utility (Using the BCP utility and CSV files) can be used.

An example of use of this utility is shown below:

```
[erwin@fedora sybase]$ isql -U sa -S LOCALHOST -D testdb -P sybase -J iso_1
1> select * from results
go
loca  locb  shipment
---------- ---------- -------------------
seattle  new-york  50.000000
seattle  chicago  300.000000
seattle  topeka  0.000000
san-diego  new-york  275.000000
san-diego  chicago  0.000000
san-diego  topeka  275.000000
(6 rows affected)
1> quit
[erwin@fedora sybase]$ cat bcp.fmt
10.0
4
1 SYBCHAR 0 0 "\n" 1 loca
2 SYBCHAR 0 10 "","" 1 loca
3 SYBCHAR 0 10 "," 2 locb
4 SYBCHAR 0 17 
"n" 3 shipment
[erwin@fedora sybase]$ bcp testdb..results out res.txt -S LOCALHOST -U sa -P sybase -J iso_1 -f bcp.fmt
Starting copy...
6 rows copied.
Clock Time (ms.): total = 1  Avg = 0 (6000.00 rows per sec.)
[erwin@fedora sybase]$ cat res.txt
"seattle","new-york",50.0
"seattle","chicago",300.0
"seattle","topeka",0.0
"san-diego","new-york",275.0
"san-diego","chicago",0.0
"san-diego","topeka",275.0
[erwin@fedora sybase]$ 
```

Note: the first column in the format file is a dummy (it has length 0). This is in order to write the leading quote, as bcp only allows for termination symbols.

This can be automated using the following GAMS code:

```gams
sets
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /

$onecho > bcp.fmt
10.0
4
1 SYBCHAR 0 0 "\n" 1 loca
2 SYBCHAR 0 10 "","" 1 loca
```
4.18.3.7.1.2 Import using the SQL2GMS utility  The SQL2GMS tool uses ADO or ActiveX Data Objects to extract data from relational databases. It can connect to almost any database from any vendor as it supports standards like ODBC. See documentation for more information.

4.18.3.7.1.3 Import using a 'SQL2GMS' VBS script  The following GAMS code will generate and execute a script written in VBScript. It mimics the behavior of SQL2GMS.EXE and can be used for debugging or the script can be passed on to the IT support people in case there are problems with accessing the database.

```gams
$ontext
  This script mimics SQL2GMS.
  Erwin Kalvelagen
  November 2006
$offtext

$onecho > sql2gms.vbs
,
  ' parameters
  t1 = 3  ' connection timeout
t2 = 0  ' command timeout
c = "Provider=MSDASQL;Driver={SQL Server};Server=DUOLAP\SQLEXPRESS;Database=testdata;Uid=gams;Pwd=gams;"  ' connection string
q = "select * from data"  ' query
o = "output.inc"  ' the output file to be generated
b = false  ' whether to quote indices (e.g. because of embedded blanks)
,
  ' create ADO connection object
  set ADOConnection = CreateObject("ADODB.Connection")
ADOVersion = ADOConnection.Version
WScript.Echo "ADO Version: ", ADOVersion
,
  ' make db connection
  ADOConnection.ConnectionTimeout = t1
  ADOConnection.ConnectionString = c
  ADOConnection.Open
,
  ' Open file
```
set fso = CreateObject("Scripting.FileSystemObject")
set outputfile = fso.CreateTextFile(o,True)
outputfile.WriteLine "*----------------------------------------------------"
outputfile.WriteLine "* SQL2GMS/Vbscript 1.0"
outputfile.WriteLine "* Connection:"&c
outputfile.WriteLine "* Query:"&q
outputfile.WriteLine "*----------------------------------------------------"

' setup query
starttime = time
ADOConnection.CommandTimeout = t2
const adCmdText = 1
set RecordSet = ADOConnection.Execute(q,,adCmdText)

' get results
NumberOfFields = RecordSet.Fields.Count
eof = RecordSet.EOF
if eof then
   WScript.Echo "No records"
   Wscript.quit
end if

' loop through records
NumberOfRows = 0
do until eof
   NumberOfRows = NumberOfRows + 1
   Row = RecordSet.GetRows(1)
   if Numberoffields > 1 then
      s = Row(0,0)
      if b then
         s = quotestring(s)
      end if
      Outputfile.Write s
   end if
   for i=2 to NumberOfFields-1
      s = Row(i-1,0)
      if b then
         s = quotestring(s)
      end if
      Outputfile.Write ".".
   Outputfile.Write s
   next
   s = Row(NumberOfFields-1,0)
   OutputFile.Write " "
   OutputFile.WriteLine s
   eof = RecordSet.EOF
loop
474

User's Guide

OutputFile.Close
Wscript.echo "Records read:"&NumberOfRows
Wscript.echo "Elapsed time:"&DateDiff("s",starttime,time)&" seconds."

function quotestring(s)
has_single_quotes = false
has_double_quotes = false
needs_quoting = false
’
’ check input string for special characters
’
for j=1 to len(s)
ch = Mid(s,j,1)
select case ch
case "’"
has_single_quotes = true
case """"
has_double_quotes = true
case " ","/",";",","
needs_quoting = true
case else
k = asc(ch)
if (k<=31) or (k>=127) then
needs_quoting = true
end if
end select
next
’
’ check if we have if gams keyword
’
kw = array("ABORT","ACRONYM","ACRONYMS","ALIAS","BINARY","DISPLAY","ELSE",
"EQUATION","EQUATIONS","EXECUTE","FILE","FILES","FOR","FREE",
"IF","INTEGER","LOOP","MODEL","MODELS","NEGATIVE","OPTION",
"OPTIONS","PARAMETER","PARAMETERS","POSITIVE","PROCEDURE",
"PROCEDURES","PUT","PUTCLEAR","PUTCLOSE","PUTHD","PUTPAGE",
"PUTTL","SCALAR","SCALARS","SEMICONT","SET","SETS","SOS1",
"SOS2","TABLE","VARIABLE","VARIABLES","WHILE")
if not needs_quoting then
for j = 0 to Ubound(kw)
if strcomp(s,kw(j),1)=0 then
needs_quoting = true
exit for
end if
next
end if
’
’ already quoted?
’
ch = left(s,1)
select case ch
case "’", """"
quotestring = s

_
_
_
_
_
_


4.19 Executing GAMS from other Environments

4.19.1 Some General Comments

Nowadays the object-oriented APIs are the most efficient and elegant way to interact with the GAMS system. They allow the effective communication of data, and do parameterized runs of GAMS. This whole chapter describes a much more basic interaction of GAMS through calling the GAMS executable from different environments. This still can be useful, e.g. if no object-oriented API is available for the particular target language (e.g. VBA).

While the principle of calling the GAMS executable holds for all operating systems, this chapter often focuses on the Windows platform.

One of the interesting problems one faces when spawning GAMS.EXE in a Windows environment is multi-threading. If one does not take precautions, a call to Shell (VB function) or CreateProcess (Win32 API function) causes GAMS to run asynchronously: the function will return while GAMS is still running. In order to read the results one would need to wait until the GAMS job has finished. The machinery for this requires some Windows trickery, and for Visual Basic version 6 and Delphi version 4 we have implemented some small examples that illustrate how this can be done.

Another issue that needs to be addressed is that GAMS needs a place to put its scratch files. By default this is the current directory, a concept that is not always clear in a windowing environment. A good way of dealing with this is to set both the current drive and the current directory before running the GAMS job. It should be noted that GAMS needs write permission there. In the examples we use the Windows TEMP directory for this, but in a real application you may want to use a designated directory. The Windows TEMP directory is found by calling the API function GetTempPath.

4.19.1.1 The GAMS architecture

GAMS itself is a console mode application. In fact it is not a single program, but a driver program (GAMS.EXE on Windows, otherwise GAMS) that executes in turn the GAMS language compiler (GAMSCMEX) or one of the solvers. For a model with a single solve statement, GAMS.EXE will first call GAMSCMEX to compile the GAMS model. Then GAMSCMEX will start executing the internal code that was generated by the compiler. As soon as it hits the instructions belonging to a SOLVE statement it will generate model instance, and GAMSCMEX will exit. Then GAMS.EXE will spawn a solver capable of solving the model. As soon as the solver is finished, GAMS.EXE will execute GAMSCMEX again so it can read the solution and can continue with executing instructions.
4.19.2 Spawning GAMS from VBA

Visual Basic for Applications (VBA) is a programming language that is built into most Microsoft Office applications, e.g. Excel and Access. A VBA program may include modules, which can be imported from files, i.e. in the VBA editor you can choose menu “File”→“Import File”. The GAMS distribution includes some VBA modules, which can be found from:

\[ \text{<GAMS System Directory>\apifiles\VBA\api, e.g. "C:\GAMS\win64\24.5\apifiles\VBA\api"} \]

Attention

- In order to avoid issues, it's recommended to use the latest version of the modules, i.e. the modules found in the latest GAMS release.

For example, the following modules can be found:

- gamsglobals.bas: Global constants that are used in other modules
- gdxvba.bas: GAMS Data Exchange Object
- idxvba.bas: GAMS IDX Object
- optvba.bas: GAMS Option Object

For more information, see Expert-Level APIs. VBA programs that use the API can be found from:

\[ \text{<GAMS System Directory>\apifiles\VBA, e.g. "C:\GAMS\win64\24.5\apifiles\VBA"} \]

The models can also be retrieved via GAMSIDE -> Model Libraries -> GAMS Data Utilities Models.

4.19.2.1 Spawning GAMS from Excel

Calling GAMS out of Excel requires some more work than just Data Exchange with Microsoft Excel. The application calling GAMS out of Excel has to:

1. Locate the GAMS system directory and adjust the system path accordingly.
2. Copy the GAMS model into a temporary directory (by default the temporary directory of Windows)
3. Extract the model data from the spreadsheet into a GAMS readable format (gdx)
4. Execute GAMS (solve the model, write the solution back to gdx file)
5. Import the model results back from the gdx files into the spreadsheet
6. Update the spreadsheet (graphics, tables)

Using Visual Basic for Applications (VBA) this can be implemented with a few lines of code:

```vba
Sub solve()
    Dim WorkDir As String
    WorkDir = TempDir()
    Call ClearSolution
    If (Not AddGAMSPath()) Then ' needed to find gdxdclib64.dll and gams.exe
        Exit Sub
    End If
    Call ExportGDXFile(WorkDir)
    Call WriteGAMSModel(WorkDir & "portfolio.gms")
    Call RunGAMS(WorkDir & "portfolio.gms", WorkDir)
    Call ReadListing(WorkDir & "portfolio.lst")
    Call ImportGDXFile(WorkDir)
End Sub
```

For further details, please inspect the VBA part of the examples below.
4.19.2.2 A Simple Example

This very simple example shows how GAMS can be invoked from an Excel spreadsheet. The "example spreadsheet" has a button, which will cause GAMS to run the trnsport.gms model stored in c:/tmp. There is no data exchange.

A more complete application will write an include file for a GAMS model, and will import a comma delimited file with results when the run is finished. An example of such a complete application is described in http://www.gams.com/mccarl/excelgams.pdf.

4.19.2.3 Sudoku Example

This "spreadsheet" is a complete example that uses GDX files to exchange information solves a 25x25 Sudoku problem using CPLEX. You will need a GAMS/CPLEX license to be able to run the spreadsheet. The MIP model solves very easily: the solution is found in the presolve phase.

Note: This spreadsheet requires distribution 22.6 or younger to work properly. If you are using an older distribution, please download "this one".

4.19.2.3.1 Problem

4.19.2.3.2 Solution

4.19.2.4 Efficient frontier example

This "example" (screen shot below) solves a series of NLPs to create an efficient frontier of a portfolio optimization problem.

Note: This spreadsheet requires distribution 22.6 or younger to work properly. If you are using an older distribution, please download "this one".

4.19.3 Spawning GAMS from C

The example below shows an absolute minimal version of calling GAMS from a C program:

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    system("gams trnsport lo=2");
    return 0;
}
```

As can be seen the GAMS executable gams.exe is called via a simple invocation of system(). The extra command line parameter lo=2 indicates that the log file is not written to the screen (this is the default) but to a file model.log.

If we want to generate some data from the application program (e.g. parameter DEMAND), we can use an include file which is written by the application. Of the way back results can be read from a PUT file. The GAMS file can now look like:
Sets
i canning plants / seattle, san-diego /
j markets / new-york, chicago, topeka /

Parameters
a(i) capacity of plant i in cases
/ seattle 350
   san-diego 600 /
b(j) demand at market j in cases
$/include demand.inc /
$include demand.inc /

Table d(i,j) distance in thousands of miles
new-york | chicago | topeka  
seattle   | 2.5     | 1.7     | 1.8
san-diego | 2.5     | 1.8     | 1.4

Scalar f freight in dollars per case per thousand miles /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
x(i,j) shipment quantities in cases
z total transportation costs in thousands of dollars ;

Positive Variable x ;

Equations
cost define objective function
supply(i) observe supply limit at plant i
demand(j) satisfy demand at market j ;
cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;
Solve transport using lp minimizing z ;

Display x.l, x.m ;

file fout /results.txt/;
put fout;
loop((i,j),
   put x.l(i,j):17:5/;
);
putclose;

The corresponding C program could look like:

```
//
// example of running a GAMS model
```
4.19 Executing GAMS from other Environments

4.19.4 Spawning GAMS from Visual Basic

This example shows how GAMS can be invoked from a VB program.

When executed a simple window appears where a GAMS model can be specified and possible other command line options. The [GAMS] button will execute the model.

When the Normal display is used, a console window will be opened. This console window will be closed automatically at the end of the run if the Close process window on completion check box is checked. Minimized does not show a window, but in the taskbar an icon will appear. If the job takes a long time, the user can click on this icon to make the console window visible. The Hidden option prevents any GAMS associated window or icon to appear.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned.

The main program is attached below, the full source for this example is "available for download".

```vb
Attribute VB_Name = "vBGams"
Option Explicit
' ------------------------------
' Module Name: VBGams
' File: gams.bas
' Version: 1.1
',
', ' Functions:
' VB_Gams32 - sets up process directory, parameter file, and console, and manages
```
4.19.5 Spawning GAMS from Delphi

This is a Delphi 4 application that has similar features as the Visual Basic application described in the previous paragraph.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned. The main code is attached below, the full source and the executable for this example are "available for download".
unit main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TForm = class(TForm)
  FileEdit: TEdit;
  FileOpenDialog: TOpenDialog;
  filelabel: TLabel;
  BrowseButton: TButton;
  CmdLineOptionsEdit: TEdit;
  CmdLineLabel: TLabel;
  RunButton: TButton;
  ReturnLabel: TLabel;
  ConsoleComboBox: TComboBox;
  ConsoleLabel: TLabel;
  CloseConsoleButton: TButton;
procedure TForm.BrowseButtonClick(Sender: TObject);
procedure TForm.RunButtonClick(Sender: TObject);
procedure TForm.Create(Sender: TObject);
procedure TForm.CloseConsoleButtonClick(Sender: TObject);
private
  { Private declarations }
  function GSExec(const ProgName,ProgParams: string;
    const wShowWindow : word;
    var rc: integer): integer;
function ExecuteGAMS(const ProgName,ProgParams: string;
  wShowWindow : word;
  AutoClose : boolean;
  var rc: integer): integer;
public
  { Public declarations }
end;

var
  MainForm: TForm;

implementation

{$R *.DFM}

function TForm.GSExec(const ProgName,ProgParams: string;
  const wShowWindow : word;
  var rc: integer): integer;
{ execute program: }
  Result: error code for starting the program
  rc : error code returned by the program
begin
  Command : String;
  ProcessInformation : TProcessInformation;
  StartupInfo : TStartupInfo;
  exitcode : dword;
  // Initialise the startup information to be the same as that of the
  // calling application. This is easier than initialising the many
  // individual startup information fields and should be fine in most
  // cases.
  GetStartupInfo(StartupInfo);
  // StartupInfo.wShowWindow determines whether the called application
  // will be initially displayed normal, maximised, minimised or some
  // other subtle variations
  StartupInfo.wShowWindow := wShowWindow;
  Command := ProgName + ' ' + ProgParams;
  if not CreateProcess(
    nil,                   {ApplicationName}
    PChar(Command),       {lpCommandLine}
    nil,                  {lpProcessAttributes}
    nil,                  {lpThreadAttribute}
    false,                {bInheritedHandles}
    NORMAL_PRIORITY_CLASS, {dwCreationFlags}
    nil,                  {lpEnvironment}
    nil,                  {lpCurrentDirectory}
    StartupInfo,          {lpStartupInfo}
    ProcessInformation    {lpProcessInformation}
  )
  then begin
    rc := 0;
    Result := GetLastError {failed to execute}
  end
  else begin
    with ProcessInformation do begin
      WaitForSingleObject(hProcess,INFINITE);
      GetExitCodeProcess(hProcess,exitcode);
      CloseHandle(hThread);
      CloseHandle(hProcess);
    end;
    rc := exitcode;
    Result := GetLastError {error code returned by the program}
  end;
end;
function TMainForm.ExecuteGAMS(
    const ProgName, ProgParams: string;
    wShowWindow: word; [ SW_NORMAL, SW_HIDE or SW_SHOWMINIMIZED ]
    AutoClose: boolean; [ close console at end ]
    var rc: integer): integer;
var ok: bool;
begin
    if AutoClose then begin
        result := GSExec(ProgName, ProgParams, wShowWindow, rc);
        exit;
    end;
    ok := AllocConsole();
    result := GSExec(ProgName, ProgParams, wShowWindow, rc);
    if (ok) then
        CloseConsoleButton.Enabled := true;
end;
procedure CdTemp;
const maxpath=260;
var path: string;
begin
    setlength(path, maxpath);
    GetTempPath(maxpath, Pchar(path));
    path := ExpandFileName(path);  // just to be sure
    ChDir(path);  // will also change drive
end;
procedure TMainForm.BrowseButtonClick(Sender: TObject);
begin
    if FileOpenDialog.Execute then
        FileEdit.Text := FileOpenDialog.Filename;
end;
procedure TMainForm.RunButtonClick(Sender: TObject);
var
    rc: integer;  // return code from GAMS.EXE
    result: integer;  // return code from GsExe
    params: string;  // command line params for GAMS.EXE
    s: string;  // for assembly of error messages
    wShowWindow: word;
    AutoClose: boolean;
begin
    CdTemp;
    extract command line parameters from edit controls
    there may be blanks in the filenames, so add quotes
    params := '"' + FileEdit.Text + '" ' + CmdLineOptionsEdit.Text;
    get wShowWindow information
    case ConsoleComboBox.ItemIndex of
        0, 1: wShowWindow := SW_NORMAL;
        2: wShowWindow := SW_SHOWMINIMIZED;
        3: wShowWindow := SW_HIDE;
        end;
    AutoClose := not (ConsoleComboBox.ItemIndex = 1);
    inform user GAMS is running
    ReturnLabel.Caption := 'Running GAMS...';
    ReturnLabel.Font.Color := ClGreen;
    Refresh;
    run GAMS.EXE
    result := ExecuteGAMS('gams.exe', params, wShowWindow, AutoClose, rc);
    check for results
    if (result <> 0) then begin
        str(result, s);
        ReturnLabel.Caption := 'Exec failed: result = ' + s;
        ReturnLabel.Font.Color := ClRed;
    end else if (rc <> 0) then begin
        str(rc, s);
        ReturnLabel.Caption := 'GAMS failed: rc = ' + s;
        ReturnLabel.Font.Color := ClRed;
    end else begin
        ReturnLabel.Caption := 'OK';
        ReturnLabel.Font.Color := ClBlack;
    end;
end;
procedure TMainForm.FormCreate(Sender: TObject);
begin
    ConsoleComboBox.ItemIndex := 0;
end;
procedure TMainForm.CloseConsoleButtonClick(Sender: TObject);
4.19 Executing GAMS from other Environments

4.19.6 Spawning GAMS from Visual C++

In "this example" we call GAMS from a simple Visual C++ application.

The additional concern is here that we want to intercept the GAMS screen output so it can be written to a multi-line edit control. For more information about the used technique see the Microsoft publication: HOWTO: Spawn Console Processes with Redirected Standard Handles (Q190351) in the Microsoft Knowledge Base.

4.19.7 Spawning GAMS from C#

Below is a simple example:

```csharp
using System.Diagnostics;

void RunGamsModel()
{
    Process p = new Process();
    p.StartInfo.FileName = gamsexe;
    p.StartInfo.Arguments = "\" + modelname + \" LO=0 --runid=\" + runid;
    p.Start();
    p.WaitForExit();
}
```

The command line formed here is `gams modelname LO=0 --runid=xxx`. The option `LO` (logoption) will disable writing a log, and the option `--runid` passes a parameter to the GAMS model (inside the model you can access this through `%runid%`).

4.19.8 Spawning GAMS from Java

Java has a class `Runtime` that implements spawning of processes using the `Exec()` method. This is quite trivial to use in applications, but in applets the default security settings don't allow this operation (in general). The problem can be solved by loading the applet from a local drive or by using a signed applet. Similarly if the Java classes are loaded from inside a database, this operation may require additional privileges. For an example see the previous section, where GAMS is called from Oracle using a Java Stored Procedure. The relevant code may look like:

```java
import java.io.File;

class RunGAMS {
    public static void main(String[] args) {
        System.out.println("Start");
        String[] cmdArray = new String[5];
        cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams";
        cmdArray[1] = "<PATH>" + File.separator + "transport.gms";
        cmdArray[2] = "WDIR=<PATH>" + File.separator + "TMP";
        cmdArray[3] = "SCRDIR=<PATH>" + File.separator + "TMP";
        cmdArray[4] = "LO=2";
        try {
            Process p = Runtime.getRuntime().exec(cmdArray);
            p.waitFor();
        } catch (java.io.IOException e) {
            System.err.println(">>>>" + e.getMessage());
            e.printStackTrace();
        } catch (InterruptedException e) {
            System.err.println(">>>>" + e.getMessage());
            e.printStackTrace();
        }
        System.out.println("Done");
    }
}
```
Below another example, which avoids problems, if the model has a long screen log (the buffer gets filled and locks the execution), which could happen if GAMS does not write the log to the file like above. This example is based on suggestion made by Edson Valle.

```java
import java.io.File;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class RunGAMS {
    public static void main(String[] args) {
        System.out.println("Start");
        String[] cmdArray = new String[5];
        cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams.exe";
        cmdArray[1] = "" + File.separator + "transport.gms";
        cmdArray[2] = "" + File.separator + "tmp";
        cmdArray[3] = "LO=3";
        try {
            Process p = Runtime.getRuntime().exec(cmdArray);
            BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String s = null;
            while((s=stdInput.readLine()) !=null){
                System.out.println(s);
            }
            p.waitFor();
        } catch (java.io.IOException e ) {
            System.err.println(">>>>" + e.getMessage() );
            e.printStackTrace();
        } catch (InterruptedException e ) {
            System.err.println(">>>>" + e.getMessage() );
            e.printStackTrace();
        }
        System.out.println("Done");
    }
}
```

For an example of GAMS usage from a Java based server environment see Alexander Sokolov, Information environment and architecture of decision support system for nutrient reduction in the Baltic Sea, Department of Systems Ecology, Stockholm University.

### 4.19.9 Spawning GAMS from a Web Server

Running GAMS remotely using a Web based thin-client architecture requires that GAMS is executed directly or indirectly from the Web server or HTTP server. A simple way of doing this is via a CGI process. Common Gateway Interface (CGI) programs can be written in many languages such as C, Perl or Delphi. CGI is relatively slow, as for each interaction, even the most simple one, a process needs to be started. Alternatives exist in the form of CGI extensions such as FastCGI or using DLLs or shared libraries. A basic algorithm for a CGI script could be:

- Create a unique directory, and CD to that directory.
- Get information from user forms and save it as GAMS readable files.
- Run GAMS making sure it does not write the log to the screen (i.e. use the option LO=2 (log to a file) or LO=3 (log to stdout)).
- Let the model write solution info to text files using the PUT statement.
- Read solution, and create formatted HTML to send back to the user.
- Remove temp files and directory.
Complications arise when there is a need to show graphics (files need to be stored somewhere and discarded after a while), when jobs take a long time to finish (you will need to add a facility where the user can pick up results at a later moment) or when the server resources can be exhausted (e.g. because of a large number of simultaneous users or because of large models).

An important complicating issue in the above list are jobs that take more time to finish: web servers like to respond to the user within a short time, and time out errors will occur if a job takes a long time. The solution is to use a queue based approach. An actual implementation available is the NEO Server — there is also a GAMS interface to NEOs: KESTREL - Remote Solver Execution on NEO Servers.

### 4.19.10 Spawning GAMS from PHP

A minimal example

**index.html**

```html
<html>
<body>
<form action="calling_gams.php" method="post">
    Solve transport with f = <select name="f">
        <option>70</option>
        <option>80</option>
        <option>90</option>
        <option>100</option>
    </select>
    <input type="submit" value="call GAMS"/>
</form>
</body>
</html>
```

**calling_gams.php**

```php
<?php
$f = $_POST['f'];
//some model data
$modelfile = 'transport_php.gms';
$city = array('new-york', 'chicago', 'topeka');
$demand = array(325.0, 300.0, 275.0);
//write demand.inc
$fh = fopen('./demand.inc', 'w+');
for($i=0; $i<count($city); $i++){
    fwrite($fh, $city[$i]." ".round($demand[$i], 0)."n");
}
fclose($fh);
//write f.inc
$fh = fopen('./f.inc', 'w+');
fwrite($fh, $f);
fclose($fh);
//call gams
system('./path/to/gams/gams ' . $modelfile . ' lo=2');
//read solutions
$fh = fopen('./results.txt', 'r');
echo "<p>result of \'.$modelfile.' (f='.$f.') :</p>";</b></p>
while (!feof($fh)){
    $line = fgets($fh);
    echo '<p>'.$line.' </p> ;
}
fclose($fh);
?>
```

**transport_php.gms**

Sets

```
Sets
i  canning plants / seattle, san-diego /
j  markets / new-york, chicago, topeka /;
```

Parameters

```
Parameters
a(i)  capacity of plant i in cases
/  seattle 350
    san-diego 600 /
```
4.20 Using GAMS Studio

This tutorial on using GAMS Studio was written initially by and is distributed by courtesy of Bruce A.
McCarrl. Bruce has extensive experience with GAMS, both as a modeler and an educator, and many

2Specialist in Applied Optimization, Distinguished Professor of Agricultural Economics, Texas A&M University, Principal, McCarl and Associates, mccarl[at]tamu.edu, brucemccarl[at]gmail.com, http://agecon2.tamu.edu/people/faculty/
GAMS users know, use, and benefit from his work. This tutorial presents a guided tour to the usage of GAMS Studio and on some items it reflects Bruce’s personal view and preferences.

4.20.1 What Is It?

*A multi platform (not only PCs with Windows) graphical interface to run GAMS*

GAMS uses two phases.

1. **Text edit**
   First, one uses a text editor and creates a file which contains GAMS instructions.

2. **File submission**
   Second, one submits that file to GAMS which executes those instructions causing calculations to be done, solvers to be used and a solution file of the execution results to be created.

Approaches to use GAMS.

1. More batch oriented traditional method use a text editor set up the model then use **DOS (or UNIX)** command line instructions to find errors in and run the model.

2. GAMS STUDIO or GAMS IDE alternatives. Graphical interfaces to create, debug, text edit and run GAMS files.

Summary of steps to using

1. Install GAMS and STUDIO on your computer

2. Open STUDIO through the icon

3. Open or create a file by going to the file selection in the upper left corner.

4. Prepare the file so you think it is ready for execution

5. Run the file with GAMS by clicking the run button or pressing F9

6. Open and navigate around the output
4.20.2 Installation

Install GAMS and STUDIO

GAMS STUDIO is automatically installed when GAMS is installed and you get the choice of whether to use STUDIO or the IDE as primary editor. To install do the following steps:

1. Download the GAMS installation file from www.gams.com under Resources and Download.

2. Start the installation.

Installer should place an icon on the desktop yielding

4.20.3 Using STUDIO after Installation

During the install you will find
Which allows you to choose which graphical interface to use as the default. For now choose STUDIO.

Open STUDIO through the icon

Open the File menu choice. Select a file.
Note, the file location determines where all generated files are placed (to place files elsewhere use the save as dialogue) and where GAMS looks for files when executing. I recommend that you make sure your file is in a place where you want it also making sure that place contains all associated files ie this is the file storage directory for this endeavor.

Fix defaults and suppress cursor driven object highlighting. When first opening Studio go to settings and I would make sure check marks are as follows especially the checked ones in the red boxes and the unchecked green one.
Here selecting the **Process Log** and possibly the **Explorer** are of interest.
The Explorer will open automatically when a file is opened and the Process Log will open when a solve is done. I usually hide the Explorer and sometimes want to get rid of the Process window.

4.20.3.1 Getting Started

4.20.3.1.1 Create or open an existing GMS file  Several cases are possible

1. Create a new file (covered later)

2. Open an existing file – at first this will be in

then you see something like
3. Open the GAMS model library explorer under the GAMS tab or press F6 (this is the simplest for first time users and the one we will use)

Select a model like transport.gms
It will be automatically saved in the last directory you used or in the STUDIO default Workspace directory.

4.20.3.1.2 Prepare file for execution When using model library `transpgms` should now appear as part of your STUDIO screen.

The STUDIO contains a full-featured editor. Go through the file and change what you want.

You can also customize the appearance somewhat through the settings choice under the File menu.
There you can change font size (under editor), tabs, default file location (called workspace) and some other features on how GAMS runs. For now leave it alone.

4.20.3.1.3 Make sure right file is executed

Unlike a lot of other programs, STUDIO does not necessarily execute the file you have been working on rather executing what it calls the "main" file.

In particular, if you activate the explorer view then the main file is the one marked with a **green triangle** as below. If the triangle is not marking the file to be run you need to right click on that file and then choose **Set as main file**. In turn the triangle will move.
4.20.3.1.4 Run GAMS by clicking the run button

The **Process Log** window (green) will then appear which logs the steps executed in running the model. The **LST file** (red) and the **navigation window** (magenta) also appear. Note to save space I would close the Explorer window and drag the **Process Log** window (green) to the bottom or the right hand side. The process window on the bottom is illustrated here and it is on the right side on the following picture.
4.20 Using GAMS Studio

Here it is after reconfiguring by dragging the Process Log up and to the right then resizing and repositioning.

Now use the process window (green) to navigate the LST file or open created files. By double clicking on lines in that window you access the LST file at particular locations or open created files. What and where you access is determined by the color of the line you click on.
### Color of Line and Function and Destination When Double Clicked

<table>
<thead>
<tr>
<th>Color of Line</th>
<th>Function and Destination When Double Clicked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue line</td>
<td>Opens LST file and jumps to corresponding place.</td>
</tr>
<tr>
<td>Non-bolded black line</td>
<td>Control click on this takes you to LST file. Double click on error message takes you to spot in LST file.</td>
</tr>
<tr>
<td>Red line</td>
<td>Marks errors encountered. Cursor is positioned at first red line. When clicked on cursor jumps to Source (GMS) file location of error. Control click takes you to error location in LST file. Error description text is in execution process log and in LST file. You can use a little red icon in source to jump to error message in LST file or a green arrow in the LST to go to error location in source.</td>
</tr>
<tr>
<td>Green line</td>
<td>Causes studio to open a file that was created during the run like a GDX, PUT or REF file.</td>
</tr>
</tbody>
</table>

---

**4.20.3.1.5 Navigate with process window**  
After double-clicking on any non-red line, or if we used the automatically open list file option our main editing window is augmented by the **LST file** (green) (see the magenta tabs)
4.20 Using GAMS Studio

We can navigate as we would with an editor or word processor, as we are automatically in STUDIO text editor.

Sometimes I want to get the process window and explorer out of the way. I can do this through the view window and also through the icons (red) on the right side of the top bar.

Clicking on these opens and closes the explorer and the process window.

4.20.3.1.6 Navigate with LXI window

<table>
<thead>
<tr>
<th>Navig. Line</th>
<th>Function and Destination When Clicked.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation</td>
<td>Jumps to top of echo print in LST file</td>
</tr>
<tr>
<td>Error Mess.</td>
<td>Jumps to list of error messages when errors present</td>
</tr>
<tr>
<td>Navig. Line</td>
<td>Function and Destination When Clicked.</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Equat. listing</td>
<td>Jumps to list of equation contents</td>
</tr>
<tr>
<td>Equation</td>
<td>Expandable allowing jump list of equation contents</td>
</tr>
<tr>
<td>Variab. listing</td>
<td>Jumps to list of variable</td>
</tr>
<tr>
<td>Variable</td>
<td>Expandable allowing jump to list of variable contents</td>
</tr>
<tr>
<td>Model stat.</td>
<td>Jumps to model statistics part of LST file</td>
</tr>
<tr>
<td>Solut.Report</td>
<td>Jumps to model summary solution report</td>
</tr>
<tr>
<td>SolEQU</td>
<td>Expandable allowing jump solution for each equation</td>
</tr>
<tr>
<td>SolVAR</td>
<td>Expandable allowing jump to solution for variable</td>
</tr>
<tr>
<td>Execution</td>
<td>Jumps to beginning of post solve execution</td>
</tr>
<tr>
<td>Display</td>
<td>Jump to displays of specific parameters and other items</td>
</tr>
</tbody>
</table>

### 4.20.3.2 Working with your own files

Now you are ready to work with your own files. You may already have a file or you may need to create one.

**Creating a new file - Two principal ways**

1. Open an existing file through the library or from a location on your hard disk. Then use the `save as` dialogue from the `file` menu to change its name. Now modify contents to what you want. You may cut and paste as in other Windows programs.

2. Open the `file` menu and use the `new` option. You will then be prompted for a name (I called it mine) with the extension `.gms` and then have a `blank screen` (green) into which you may type GAMS instructions.

Save that file wherever you want and reopen it that will become the new default directory (this only occurs upon an open not a save as) and any subsequent non library opens, simple includes or saves will go there. Imports from the library go to the default workspace.
4.20.4 The Welcome and Explorer Windows

As you have worked with STUDIO you will find it shows a Welcome (magenta) window (unless you told it not to in settings), that remembers previous files plus gives access to documentation.

There as of now you get links to files recently used, and links to some documentation.

Also on the left hand side is an Explorer (green) window that shows recent files and allows you to open by double clicking. This remains whether you are in the welcome page or editing files unless it is not checked in the View dialogue.

4.20.5 Fixing Compilation Errors

No one is perfect, errors always occur. The STUDIO can help you in finding and fixing those errors.

Let’s look at an example with a small typo to illustrate (note: source vs. sorce). A run yields the windows below where the right part is the process window (green) positioned to show the first error and the center is the lst file (magenta) positioned to show the first error.
The red lines in the process window mark errors with error messages just below. To see where the errors occurred click on the very first red line which opens the source file or look at the LST file where the marking $ appears.

A click on the red line takes you to the place in the source where the error was made. The tip here is always start at the top of the process file when doing this.

A double click on a black line takes you to the error location in the LST file.

Also in the source file window note the small red circle that appears on the line with the error. Clicking on that transfers you to the LST file where the error message appears.

And there a small gray circle with an arrow in it that when clicked on will transfer you back to the source file.
4.20.6  Ways to find and/or replace text strings

For finding you can use the search menu under the Edit tab or control F

Subsequently you get
Which allows both search and replace. As of now this opens with the last option you used and may start with initially with an all files setting.

After the search you find

Where all occurrences of the search term in the current file are highlighted in yellow and to the right there is a list of all files that contain that search term (green) in all of the directories corresponding to files opened in the explorer.

The search dialogue is straight forward with arrow keys to jump to next (F3 also works for forward and shift F3 for backward) and the all files can be manipulated as shown (in yellow).

### 4.20.7 Matching Parentheses

The STUDIO provides you with a way of checking on how the parentheses match up in your GAMS code. This involves positioning your cursor just before an open parentheses or just after a close (Just after also works if there are not multiple parentheses). In either case the corresponding close or open is highlighted in green. Additionally positioning the cursor right before the beginning parentheses and tapping Ctrl + b will lead you to the matching ending parentheses whether it be 1, 100, or 1000+ lines away and vice versa.
When the expression in parentheses carries on the several lines the marking is in multiple lines and Ctrl + b will transfer position back and forth.

This feature will also match up { } or [ ] with a red marking if the types don’t match.

### 4.20.8 Moving Blocks

The STUDIO allows one to move text blocks through standard Cut and Paste Operations in two fashions.

1. One can identify a continuous block of text with the mouse or the keyboard (the latter involves putting the cursor at a beginning point then hold the shift key down and use the arrow keys).

2. One can identify a **column block of text** (blue) with the mouse or the keyboard (these involve holding **alt and shift down** then moving the mouse or the cursor with the arrow keys).

<table>
<thead>
<tr>
<th>Table</th>
<th>d(i,j) 'distance in thousands of miles'</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-york</td>
<td>chicago</td>
</tr>
<tr>
<td>seattle</td>
<td>2.5</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
</tr>
</tbody>
</table>

In turn copy, cut, and paste can be done with the Edit menu or with control c, x and v respectively as in normal windows. Control insert also pastes.

Also if one types with a column block highlighted than the character goes in all rows of the block and replaces what was there before.
4.20.9 Syntax Coloring

A feature in the STUDIO is syntax coloring. The STUDIO recognizes a subset of the GAMS syntax and reflects this in the display colors. Note in the display below that commands, explanatory text and set elements are differentially colored.

```
Parameter
  a(i) 'capacity of plant i in cases'
    / seattle 350
    / san-diego 600 /
  b(j) 'demand at market j in cases'
    / new-york 325
    / chicago 300
    / topeka 275 /;
Table d(i,j) 'distance in thousands of miles'
  new-york  chicago  topeka
  seattle    2.5    1.7    1.8
  san-diego  2.5    1.8    1.4;
Scalar f 'freight in dollars per case per thousand miles' / 90 /;
Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;
Variable
  x(i,j) 'shipment quantities in cases'
  z    'total transportation costs in thousands of dollars';
Positive Variable x;
Equation
  cost    'define objective function'
  supply(i) 'observe supply limit at plant i'
  demand(j) 'satisfy demand at market j';
  cost..  z =e= sum((i,j), c(i,j)*x(i,j));
```

For now unlike in the IDE one cannot alter these syntax colors.
4.20.10 Showing where a symbol appears

Studio has a feature that identifies where a symbol is used. Taking the library file *agreste.gms* and positioning the cursor over a symbol like c (blue) then double clicking places a (gray) highlight over all visible instances of that symbol.

```
$Title Model Definition

Variable
  xcrop(p,z) 'cropping activities' (ha),
  xliver(z)  'livestock activity defined on feed techniques' (head),
  xlive     'livestock production' (head),
  lswitch(s) 'land downgrading' (ha),
  xprod(s)  'crop production' (ton),
  cons(dr)  'on-farm consumption' (ton),
  sales(s)  'crop sales' (ton),
  flab(tm)  'family labor' (man-days),
  tlab(tm)  'temporary labor' (man-days),
  plab      'permanent labor' (workers),
  rationr  'livestock ration requirements' (cr),
  pdev(ty)  'positive price deviations' (cr),
  ndev(ty)  'negative price deviations' (cr),
  yfarm     'farm income' (cr),
  revenue   'from crop and livestock sales' (cr),
  cropcost  'accounting: cropping activities cost' (cr),
  labcost   'accounting: labor costs - including family' (cr),
  vetcost   'accounting: veterinary services cost' (cr);

Positive Variable xcrop, xliver, lswitch, sales, cons, flab, tlab, plab,

Equation
  landb(s) 'land balance',
  lbal    'livestock balance',
  rliv    'livestock ration requirements definition',
  mbalc(s) 'material balance: crops',
  dprod(s) 'crop production definition',
  cond    'on farm consumption definition',
  labc(tm) 'labor supply-demand relation',
  ddev(ty) 'crop price deviation definition';
```

4.20.11 Accessing Documentation Via the Help

To access GAMS documentation choose help or F1
And if you choose documentation you get access through a window in lower right corner as shown. Therein you can navigate, search, and look at the index. Also if you get on the documentation page and go to the bottom you find the McCarl guide.

If your cursor is on a reserved word in the source file and you press F1 an index opens to all treatments of that word although in some strange places.

There is a document that explains features in the STUDIO and a YouTube video that can be accessed through the welcome page or in the general GAMS documentation under Tools.
4.20.12 Making GDX files

STUDIO can create and show GDX files. To create in STUDIO use the GAMS tab and choose a GDX creating choice. Alternatively use one of the other ways in the language as we will discuss elsewhere.

To browse a GDX file open it with the file open dialogue or click on green lines for ones created during a run in the process window and then for the transport example:

Here in the left-hand panel one gets an alphabetical listing of the items in the GDX file and their characteristics. Then clicking on a left-hand panel item leads to the item contents being revealed in the right hand panel. As of now you can reorder columns and filter values then in table mode can move things in rows into columns and vice versa.
4.20.13 Examining GDX files

When you have opened a GDX file and selected an item then you get a window like that on the previous page.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Name</th>
<th>Type</th>
<th>Dim</th>
<th>Records</th>
<th>Text</th>
<th>i</th>
<th>j</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>a</td>
<td>Parameter</td>
<td>1</td>
<td>2</td>
<td>capacity of plan</td>
<td>seattle</td>
<td>new-york</td>
<td>2.5</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>Parameter</td>
<td>1</td>
<td>3</td>
<td>demand at mar</td>
<td>seattle</td>
<td>chicago</td>
<td>1.7</td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>Parameter</td>
<td>2</td>
<td>0</td>
<td>transport cost i</td>
<td>seattle</td>
<td>topeka</td>
<td>1.8</td>
</tr>
<tr>
<td>10</td>
<td>cost</td>
<td>Equation</td>
<td>0</td>
<td>1</td>
<td>define objective</td>
<td>san-diego</td>
<td>new-york</td>
<td>2.5</td>
</tr>
<tr>
<td>12</td>
<td>demand</td>
<td>Equation</td>
<td>1</td>
<td>0</td>
<td>satisfy demand</td>
<td>san-diego</td>
<td>chicago</td>
<td>1.8</td>
</tr>
<tr>
<td>5</td>
<td>di</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>distance in thou</td>
<td>san-diego</td>
<td>topeka</td>
<td>1.4</td>
</tr>
</tbody>
</table>

If in the right-hand panel you dragged \textbf{i to the right of value} (brown below) you get

Also by \textbf{clicking on the header} you \textbf{cause the array to be sorted with a ^ marking the sort order} and by \textbf{clicking on}
you can establish a filter (green above and brown below) and choose what elements to display (blue below).
4.20.14 A difficulty you will have

When using and teaching both the IDE and STUDIO, I find that file locations sometimes gives me fits. I have a rule of thumb to avoid problems, but you will most likely inadvertently not follow it.

When working with a model with multiple files make sure they are located in the same directory location.

You do not have to follow this rule but deviations are the same as asking for trouble. When you are actively working on a file that you have opened from a directory then STUDIO will look for files in the directory where active file is located.

Also when you run a file which contains includes GAMS will look in the directory where the active file is located.

4.20.15 Command Line Parameters

Experienced DOS or UNIX based GAMS users are used to having command line parameters associated with their GAMS execution commands. In STUDIO a command line parameter entry box (magenta) is available just to the right of the execute button. The parameters are associated with a file and STUDIO will remember these whenever the file is opened.
This is particularly useful for save, restart, limrow, limcol, pw and rf parameters as once they are defined they are associated with every subsequent use of the file.

As of now there is no way to provide command line parameters for use in every run (like a wide page width) as there was in the IDE.

GAMS provides assistance with command line parameters through the command line parameter editor. To activate it check the symbol at the far right (red) up top. Then part of the screen becomes as below. In the left hand part existing command line parameters (purple) in use (magenta) are listed along with their values. On the right all possible command line parameters are listed with definitions (green) and a search dialogue is provided (blue).
The search dialogue looks for text in any of the fields, thus typing in \texttt{wid} brings up page width as does \texttt{pw} and \texttt{maxi} brings up those with maximum in the description.

Note pressing F1 while the cursor is on a command line entry opens help to a place with a detailed description of that parameter.

### 4.20.16 Solver Option Files

STUDIO allows you to edit or create a solver option file. When using the solver \texttt{Conopt} you create \texttt{conopt.opt} or with \texttt{Cplex} use \texttt{cplex.opt} (or \texttt{.opt} or \texttt{.10} or \texttt{.11}) and more generally \texttt{solvername.opt}. Specifying such a file allows you to manipulate the way the solver works on the model. For example, you can alter scaling, tolerances, iteration limits, solver algorithm, and branch and bound strategy along with many other things.

Now in Studio the option file editor is accessed by either

- Right clicking on the \texttt{group name} (red) for a group where you want an option file then choosing \texttt{add new solver option file} (blue) and picking the \texttt{appropriate solver name} (magenta)

- Creating a file like \texttt{baron.opt} then when you bring that up in the editor the option file editor appears. In the editor you see \texttt{options in use} (blue), \texttt{possibilities and definitions} (red)

### 4.20.17 Using Libraries

STUDIO allows one to access libraries under the GAMS menu item and choosing the \texttt{Model Library Explorer} (purple).
4.20 Using GAMS Studio

In turn this opens a screen which identifies names of the libraries in tabs with a count of included files and, for the tab selected, the library contents. Furthermore across the top are tabs that when clicked on cause sorts. The picture below came after I clicked on the tab name with the model library active.

Users can define their own libraries. They have to define what’s called a GLB file and the formatting creation of this is discussed in my newsletter in the second issue: [https://www.gams.com/mccarl/newsletter/news2.htm](https://www.gams.com/mccarl/newsletter/news2.htm).

4.20.18 Using reference files - Listing and Unraveling Data items

GAMS has a command line parameter to create reference files (aka. .ref files) called reference (short: rf). These files give information on where things are defined and used.

Why? GAMS Modelers sometimes have to deal with complex implementations that

- Use include statements to incorporate numerous files.
- Have been developed by others.
- Have a complex structure with definitions and uses of items widely spread in a file or files.
- Contain items that are defined but never used.
- Were developed some time ago but are not extensively documented.
When faced with such cases one often asks

- Are there items defined in the program that are not used and if so what are they and where are they?
- In what files is an item defined, declared and used?

To resolve these questions some years ago I wrote a Fortran program called GAMSMAP but never made it widely available. Subsequently it was included into the GAMSIDE and then STUDIO.

When a `.ref` file is opened it creates a window as follows

![Example of GAMSMAP output](image)

The **column on the left** (red) gives tags that can be clicked on and the key ones are

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Symbols</td>
<td>a list of items that are declared and a count of them</td>
</tr>
<tr>
<td>Set</td>
<td>a list of the sets that exist in the program and a count</td>
</tr>
<tr>
<td>Variable</td>
<td>a list of the variables in the program and a count</td>
</tr>
<tr>
<td>Parameter</td>
<td>a list of the parameters (items defined in scalar, parameter or table statements) and a count.</td>
</tr>
<tr>
<td>Equation</td>
<td>a list of the equations in the program and the names of files in which they appear</td>
</tr>
<tr>
<td>Model</td>
<td>a list of the models that exist in the program and the names of files in which they appear</td>
</tr>
<tr>
<td>Unused</td>
<td>a list of items that are declared or that are not used anywhere in the model</td>
</tr>
<tr>
<td>File used</td>
<td>a list of the files included into the model</td>
</tr>
</tbody>
</table>

### 4.20.18.1 All Symbols Tab

The all symbols tab causes the output to appear as follows:
4.20 Using GAMS Studio

This shows all symbols used. The columns give

- the symbol name (Name),
- its type (Type),
- its dimension (Dim),
- the sets over which it is dimensioned (Domain)
- the explanatory text used in its declaration (Text).

Things can be sorted by clicking on the column headings.

One can search for symbols using the box at the bottom

4.20.18.2 Sets, Parameters etc. Tabs

The display contains 5 lines that when activated give information for sets, parameters etc. The resultant display lists all items falling in a class (for all things that are sets or parameters etc.) and information on their characteristics and use. Specifically one gets output as follows (for the sets in this case)
In this output once clicked on the entries tell the names of the files in which certain things happen relative to the identified items. The categories of things include the file name, line and column where items are declared, defined and used as discussed below.

Clicking on a line takes you to the file and place where the item appears.

Clicking on a column name causes it to be sorted.

After clicking on an item one gets

Entries in location columns identify files where items are

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declared</td>
<td>places where the named item is declared in a Set, Parameter, Table, Scalar, Variable, Equation, Acronym, File or Model command. This will be the first appearance.</td>
</tr>
<tr>
<td>Defined</td>
<td>places set elements or data are explicitly entered. For equations this tells where the .. specification begins.</td>
</tr>
<tr>
<td>Assigned</td>
<td>places where items appear on left hand side of an assignment statement</td>
</tr>
<tr>
<td>Implicitly-Assigned</td>
<td>places where an equation or variable has data put into it by the results of a solve statement.</td>
</tr>
<tr>
<td>Controlled</td>
<td>places where set is used in controlling a sum or defining an equation</td>
</tr>
<tr>
<td>Ref</td>
<td>places where item is on left hand side of assignment statement or in a model equation</td>
</tr>
</tbody>
</table>
Clicking on lines here opens the subject file in the specific place.

### 4.20.18.3 File used Tab

The file used tab causes the output to appear as follows:

![File Location](image)

This gives the names of the files included in the program with their full path references.

### 4.20.18.4 Unused Tab

The unused tab identifies items that are declared (in set parameter etc. statements) but are never used on the right hand side of an assignment (=) statement or in a model equation:

![Unused Tab](image)

which shows the same information as in the All Symbols tab except when an item is clicked on one gets information on the files where it is declared.
4.20.18.5 Steps to use the Reference File Viewer

The Reference File Viewer will only work after a particular “reference file” has been created by a GAMS run. The file is generated by adding the `rf` option to the command line call entering a command in the command line box of Studio as follows (a=c cause compilation only)

Once the file has been run with the `rf` option the logfile is augmented with the (green) line identifying the `.ref` file name as below

Double clicking on this opens the Reference File Viewer window.

The run of GAMS with the `rf` option places the name of the `.ref` file in green in the GAMS log file and double clicking on that line causes the Reference File Viewer to be opened.

Several notes about the run

- The `rf=` command specifies the name of the file the Reference File Viewer will use. Generally, the Reference File Viewer expects it to have the extension `.ref`.
- The `.ref` file only covers the program components in a run and does not include any information from restart files. In general it is best to explicitly use all the files in one program without use of save and restart.
- It is often useful to just generate the reference file without any execution on behalf of the GAMS program. This is done by including the `a=c` option on the command line or in the command parameter IDE box.
4.20 Using GAMS Studio

4.20.19 Editing Solver Option Files

Sometimes it is desirable to change solver specific options. This includes modifying things like branch and bound strategy, degree of scaling, barrier versus simplex solution approach, solution tolerances etc. The specific options allowed are described in the solver manuals. In addition Studio contains a solver option file editor that presents a solver specific menu of all possible options, default values and ones that have been altered. This is activated by either

- Opening or saving a file with solvername.opt (Like MINOS.opt)
- Right clicking on a group file name (red) in the Studio Explorer then selecting add new solver option file (blue) and subsequently selecting one of the many available solver names (magenta and only a few shown). This will create a file you then need to save.

Once it opens the general format of the solve option editor is as follows. Note the specific content depends on the solver chosen and in this case I use GAMSCHK.
Here in the right part we see a list of all possible options (magenta) for this solver with boxes checked for those that are in the option file. A brief description of default values (blue), entry type (red), if relevant allowable settings (yellow) and a one sentence description of the option (green). In the left part we see the current entries (purple) in the option file and their settings (purple).

Subsequently when this file is opened or its tab activated then this solver option editor appears.

Note, pressing F1 while on an option opens help to a place with a detailed description of that option.

4.20.20 When is it Not Worth Using?

There are costs and benefits of these approaches.

The STUDIO is much easier for simple models and can run across platforms.

The DOS/command line approach is generally better for models in customized environments.

A development strategy for more complex implementations

1. Use the STUDIO to get it right
2. Debug components of large models using save and restart
3. Then use script files or DOS/UNIX with batch files such as
   - GAMS mymodel -lo 0 -s ./t/save1
   - call myprogram.exe
   - GAMS moremod -lo 0 -r ./t/save1

4.20.21 What does it not do?

A number of IDE features are currently missing. Just to inventory (note the last three are pretty unimportant to me.)

- Ability to save output from the reference files
- Ability to change the file associations when clicking on them in Windows Explorer (you get more than I like at the moment and sometimes I want more)
- Project files (mixed blessing)
- Text file differencing (can use other programs)
- Spell checking

4.20.22 What does it not do so well?

Here are also some difficulties with it as of today

- Washed out colors – on my machine the syntax and error coloring is not very prominent.
- Excessive library choices and no way to put the ones you have no interest in somewhere out of the way. Plus any user defined libraries are at the end and there is no way to reorder.
4.21 Introduction

4.21.1 Summary

Today, algebraic modeling languages are widely accepted as the best way to represent and solve mathematical programming problems. Their main distinguishing features are the use of relational algebra and the ability to provide partial derivatives on multidimensional, very large and sparse structures. In this chapter we will briefly describe some of the origins of GAMS and provide background information that shaped early design decisions.

4.21.2 The Origins of GAMS

The initial Research and Development of GAMS was funded by the International Bank for Reconstruction and Development, usually referred to as The World Bank. Since 1987, further Research and Development has been funded by GAMS Development Corporation. GAMS was developed in close cooperation of mathematical economists who were and still are an important group of GAMS users. The synergy between economics, computer science and operations research was the most important success factor in the development of the system. Mathematical Programming and economics theory are closely intertwined. The Nobel Prize in Economics awarded to Leonid Kantorovich and Tjalling Koopmans in 1975 for their “contribution to the theory of optimal allocation of resources” was really a prize in mathematical programming. Other Nobel laureates like Kenneth Arrow in 1972, Wassily Leontief in 1973, and Harry Markowitz in 1990 are well known names in math programming. Another early example of this synergy is the use of LP in refining operations, which was started by Alan Manne, an economist, with his book on Scheduling of Petroleum Refinery Operations in 1956.

The origins of linear programming algorithms all go back to George Dantzig’s early work in the 1940s and 1950s. Computing technology and algorithmic theory had developed at a rapid pace. Thirty years later, we could solve problems of practical size and complexity that allowed us to test of the economic theory on real life problems. The research agenda at the World Bank in the 1970s and 1980s created the perfect environment to bring different disciplines together to apply mathematical programming to research and operational questions in Economic Development.

4.21.3 Background and Motivation

From the very beginning, the driving force behind the development of the General Algebraic Modeling System (GAMS) has been the users of mathematical programming who believed in optimization as a the powerful and elegant framework for solving real life problems in the sciences and engineering. At the same time, these users were frustrated with the high cost, skill requirements, and overall low reliability of applying optimization tools. Most of our initiatives and support for new development came from the worlds of economics, finance, and chemical engineering. These disciplines find it natural to view and understand the world and its behavior as a mathematical program.

GAMS’s impetus for development arose out of the frustrating experiences of a large economic modeling group at the World Bank. In hindsight, one may call it a historical accident that in the 1970s mathematical economists and statisticians were assembled to address problems of development. They used the best techniques available at the time to solve multisectoral economy-wide models and large simulation and optimization models in agriculture, steel, fertilizer, power, water use, and other sectors. Although the group produced impressive research, initial successes were difficult to reproduce outside their well functioning research environment. The existing techniques to construct, manipulate, and solve such models required several manual, time-consuming, and error-prone translations into the different, problem-specific representations required by each solution method. During seminar presentations, modelers had to defend the existing versions of their models, sometimes quite irrationally, because the time and money needed to make proposed changes were prohibitive. Their models just could not be moved to other environments.
because special programming knowledge was needed, and data formats and solution methods were not portable.

The idea of an algebraic approach to represent, manipulate, and solve large-scale mathematical models fused old and new paradigms into a consistent and computationally tractable system. Using matrix generators (see appendix GAMS versus Fortran Matrix Generators) for linear programs taught us the importance of naming rows and columns in a consistent manner. The connection to the emerging relational data model became evident. Painful experience using traditional programming languages to manage those name spaces naturally lead one to think in terms of sets and tuples, and this led to the relational data model. Combining multidimensional algebraic notation with the relational data model was the obvious answer. Compiler writing techniques were by now widespread, and languages like GAMS could be implemented relatively quickly. However, translating this rigorous mathematical representation into the algorithm specific format required the computation of partial derivatives on very large systems. In the 1970s, TRW developed a system called PROSE that took the ideas of chemical engineers to compute point derivatives that were exact derivatives at a given point, and to embed them in a consistent, Fortran-style calculus modeling language. The resulting system allowed the user to use automatically generated exact first and second order derivatives. This was a pioneering system and an important demonstration of a concept. However, in our opinion PROSE had a number of shortcomings: it could not handle large systems, problem representation was tied to an array-type data structure that required address calculations, and the system did not provide access to state-of-the-art solution methods. From linear programming, we learned that exploitation of sparsity was the key to solve large problems. Thus, the final piece of the puzzle was the use of sparse data structures.

With all pieces in place, all we had to do was adopt the techniques to fit into one consistent framework and make it work for large problems.

4.21.4 Design Goals and Changing Focus

The original and still valid goal is to improve the model builder’s productivity, reduce costs, and improve reliability and overall credibility of the modeling process. To achieve this, we established the following key principles to guide the GAMS development:

- The problem representation is independent of the solution method.
- The data representation follows the relational data model.
- The problem and data representations are independent of computing platforms.
- The problem and data representations are independent of user interfaces.
- Optimization methods will fail, and systems have to be designed to be fail-safe.

Another way to express these principles is to think in terms of layers of representations and capabilities that have clearly defined interfaces and functions. The oldest and most basic layer is the solver layer or implementation of a specific algorithm. Above the solver is the model layer, expressed in an algebraic modeling language. The modeling layer translates the mathematical representation into a computational structure required by a specific solution method and provides various services such as function and derivative evaluations and error recovery. Above the modeling layer is the application or domain layer, which is highly context sensitive and has knowledge about the problem to be solved and the kind of user interacting with the system.

The representation of the model in GAMS is in a form that can be easily read by humans and by machines. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past (which was a burden to maintain, and which was seldom up-to-date) is no longer needed. Moreover, the design of GAMS incorporates the following features that specifically address the user's documentation needs:
• A GAMS model representation is concise, and makes full use of the elegance of the mathematical representation.

• All data transformations are specified concisely and algebraically. This means that all data can be entered in their most elemental form and that all transformations made in constructing the model and in reporting are available for inspection.

• Explanatory text can be made part of the definition of all symbols and is reproduced whenever associated values are displayed.

• All information needed to understand the model is in one document.

Of course some discipline is needed to take full advantage of these design features, but the aim is to make models more accessible, more understandable, more verifiable, and hence more credible.

It is instructive to put the development of modeling systems into some historic perspective and see how the focus and technical constraints have changed in the last 30 years. We can observe three major phases that shift the emphasis from computational issues to modeling issues and finally the application or the real problems. Each phase defined one of the main system layers discussed above. The dominant constraints in the first phase were the computational limits of our algorithms. Problem representation had to abide by algorithmic convenience, centralized expert groups managed large, expensive and long lasting projects and end users were effectively left out. The second phase has the model in focus. Applications are limited by modeling skill, project groups are much smaller and decentralized, the computational cost are low and the users are involved in the design of the application. Applications are designed to be independent of computing platforms and frequently operate in a client-server environment.

We believe that we are entering a third phase which has the application as its focus and the optimization model is just one of many analytic tools that help making better decisions. The users are often completely unaware of any optimization model or use a mental model that is different from the actual model to solved by optimization techniques. User interfaces are build with off-the-shelf components and frequently change to adjust to evolving environments and new computing technologies. As with databases, modeling components have a much longer life than user interfaces. We have observed cases where the model has remained basically unchanged over many years, whereas the computing environments and user interfaces have changed several times. The solvers used to solve the models have changed, the computing platforms have changed, the user interfaces have changed and the overall performance of the model has changed without any change in the model representation.

4.22 GAMS Programs

4.22.1 Introduction

This chapter provides a look at the structure of the GAMS language and its components. It should be emphasized again that GAMS is a programming language, and that programs must be written in the language to use it. A GAMS program is contained in a disk file, which is usually constructed with a text editor of choice (e.g. the GAMS IDE). When GAMS is ’run’, the file containing the program (the input file) is submitted to be processed. After this processing has finished, the results, which are in the output file(s), can be inspected. By default the GAMS log appears on the screen while GAMS runs, keeping the user informed about progress and error detection. It is the responsibility of the user to inspect the output file(s) carefully to see the results and to diagnose any errors.
4.22.2 The Structure of GAMS Programs

GAMS programs consist of one or more statements (sentences) that define data structures, initial values, data modifications, and symbolic relationships (equations). While there is no fixed order in which statements have to be arranged, the order in which data modifications are carried out is important. Symbols must be declared as to type before they are used, and must have values assigned before they can be referenced in assignment statements. Each statement is followed by a semicolon except the last statement, where a semicolon is optional.

Note

The semicolon at the end of a statement can be omitted if a new GAMS keyword follows. However, to improve readability of the code, it is recommended to use the semicolon at the end of a statement anyway.

4.22.2.1 Format of GAMS Input

GAMS input is free format. A statement may be placed anywhere on a line, multiple statements may appear on a line, or a statement may be continued over any number of lines as follows:

```
statement;
statement;
statement; statement; statement;
the words that you are now reading is an example of a very long statement which is stretched over two lines;
```

Blanks and end-of-lines may generally be used freely between individual symbols or words. GAMS is not case sensitive. This means that lower and upper case letters may be mixed freely but are treated identically. Up to 80,000 characters may be placed on a line and completely blank lines may be inserted for easier reading.

Not all lines are a part of the GAMS language. Two special symbols, the asterisk ‘∗’ and the dollar symbol ‘$’ may be used in the first position on a line to indicate a non-language input line. An asterisk in column one means that the line will not be processed, but treated as a comment. For more on comments, see section Comments. A dollar symbol in the first position indicates that compiler options or directives are contained in the rest of the line (see chapter Dollar Control Options for more information).

For example, multiple files may be used as input through the use of the $include facility. In short, the statement

```
$include file1
```

inserts the contents of the specified file (file1 in this case) at the location of the call. A more complex versions of this is the option $batinclude. Both options are introduced and discussed in details in chapter Dollar Control Options.
4.22.2 Classification of GAMS Statements

Each statement in GAMS is classified into one of two groups:

1. Declaration and definition statements.
2. Execution statements.

A declaration statement describes the class of a symbol. Often initial values are provided in a declaration, then it may be called a definition. The specification of symbolic relationships for an equation is a definition. The declaration and definition statements are:

- acronym
- alias
- equation declaration
- equation definition
- file
- function
- model
- parameter
- scalar
- set
- table
- variable

Execution statements are instructions to carry out actions such as data transformation, model solution, and report generation. The execution statements are:

- abort
- assignment
- break
- continue
- display
- execute
- for
- if
- loop
- option
- put
- put_utility
Note
While an assignment is an execution statement, it also defines the symbol on the left hand side of the assignment.

Although there is great freedom about the order in which statements may be placed in a GAMS program, certain arrangements are commonly used. The two most common are discussed next.

4.22.2.3 Organization of GAMS Programs

One common style of organizing GAMS statements places the data first, followed by the model and the solution statements.

Style 1:

*Data:*

Set declarations and definitions

Parameter declarations and definitions

Assignments

Displays

*Model:*

Variable declarations

Equation declarations

Equation definitions

Model definition(s)

*Solution:*

Solve(s)

Displays
In this style of organization, the sets are placed first. Then the data is specified with parameter, scalar, and table statements. Next, the model is defined with the variable declarations, equation declarations, equation definitions and one or more model statements. Finally, the model is / models are solved and the results are displayed. One can refer to the model transport as an example for this style.

A second style emphasizes the model by placing it before the data. This is a particularly useful order when the model is to be solved repeatedly with different data sets.

**Style 2:**

*Model:*

- Set declarations
- Parameter declarations
- Variable declarations
- Equation declaration
- Equation definition
- Model definition

*Data:*

- Set definitions
- Parameter definitions
- Assignments
- Displays

*Solution:*

- Solve
- Displays

Here, there is a separation between declaration and definition. For example, sets and parameters may be declared first with the following statements:

```
Set c "crops" ;
Parameter yield(c) "crop yield" ;
```

Later they may be defined with the statements:

```
Set c / wheat, clover, beans / ;
Parameter yield(c) / wheat 1.5
                     clover 6.5
                     beans 1.0 / ;
```

The first statement declares that the identifier c is a set and the later statement defines the elements in this set. Similarly, in the second statement yield is declared to be a parameter and later the corresponding data is given.

**Note**

Sets and parameters that are used in equations must be declared before the equations are specified. However, they may be defined after the equation specifications but before the specific equation is used in a solve statement. This gives GAMS programs substantial organizational flexibility.
4.22.3 Data Types and Definitions

Each symbol or identifier has exactly one of the following basic GAMS data types:

- acronyms
- equations
- files
- functions
- models
- parameters
- sets
- variables

Note
- Scalars and tables are not separate data types but convenient input formats for the data type parameter. For details see the overview Parameters, Scalars and Tables.
- GAMS uses shorthand symbols for each data type in the output. For details see the overview GAMS Data Types and their Shorthand Symbols.

Declarations have common characteristics. The following example has a typical structure:

Parameter a(i,j) "input-output matrix" ;

The structure is:
Keyword for data type - identifier (with index list) - explanatory text ;.

Note that the index list (or domain list) and the explanatory text are always optional characteristics. However, we recommend to specify the index list if the data type is defined over a domain; the advantages of this practice are outlined in section Domain Checking.

Note
Variables, sets, parameters and equations may be declared and defined over one or more indices or dimensions. Currently the maximum number of dimensions for all these data types is 20.

It is also recommend to add an explanatory text for reasons of clarity. For more on explanatory texts, see section Text below. Other examples for declarations follow:

Set time "time periods" ;
Model turkey "turkish fertilizer model" ;
Variables x,y,z ;

Observe that in the last example a number of identifiers (separated by commas) is declared in one statement.
4.22 GAMS Programs

4.22.4 Language Items

Before proceeding with more language details, a few basic symbols need to be defined and the rules for recognizing and writing them in GAMS established. These basic symbols are often called lexical elements and form the building blocks of the language. They are:

- characters
- comments
- delimiters
- identifiers (idents)
- labels
- numbers
- reserved words and tokens
- text

Each of these items is discussed in detail in the following subsections.

Attention

As noted previously, GAMS is not case sensitive, so we may use any mix of lower and upper case.

4.22.4.1 Characters

A few characters are not allowed in a GAMS program, because they are illegal or ambiguous on some machines. Generally, all unprintable and control characters are illegal. The only place where any character is legal, is in an `$ontext-$offtext` block as illustrated in section Block Comments below. For completeness, the full set of legal characters are listed in Table 1. Note that most of the uncommon punctuation characters are not part of the language, but they may be used freely in the context of explanatory texts, comments, and labels (if quoted). Similarly, special language specific characters (e.g. ä, ß, à, é, ç, ì, ñ) may also be used freely in explanatory texts, comments, and labels (if quoted).

Table 1: Legal Characters

<table>
<thead>
<tr>
<th>Legal Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A to Z</td>
<td>alphabet</td>
</tr>
<tr>
<td>a to z</td>
<td>alphabet</td>
</tr>
<tr>
<td>0 to 9</td>
<td>numerals</td>
</tr>
<tr>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
</tr>
<tr>
<td>=</td>
<td>equals</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>( )</td>
<td>parenthesis</td>
</tr>
<tr>
<td>[ ]</td>
<td>square brackets</td>
</tr>
<tr>
<td>{ }</td>
<td>braces</td>
</tr>
<tr>
<td>’</td>
<td>single quote</td>
</tr>
<tr>
<td>“</td>
<td>double quote</td>
</tr>
</tbody>
</table>
### Legal Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>back slash</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>.</td>
<td>dot</td>
</tr>
<tr>
<td>?</td>
<td>question mark</td>
</tr>
<tr>
<td>!</td>
<td>exclamation mark</td>
</tr>
<tr>
<td></td>
<td>space</td>
</tr>
<tr>
<td>-</td>
<td>underscore</td>
</tr>
<tr>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td>^</td>
<td>circumflex</td>
</tr>
<tr>
<td>#</td>
<td>pound sign</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>%</td>
<td>percent</td>
</tr>
<tr>
<td>@</td>
<td>at</td>
</tr>
<tr>
<td>$</td>
<td>dollar</td>
</tr>
</tbody>
</table>

**Note**

Not every character listed above is allowed to be used in every place (for example, identifiers have certain limitations).

---

### 4.22.4.2 Reserved Words

GAMS, like other programming languages such as C and Java, uses reserved words (often also called keywords) that have predefined meanings. Users are in general not permitted to use these for their own definitions, neither as identifiers nor labels. The complete list of reserved words is given below. In addition, a small number of symbols constructed from non-alphanumeric characters have a meaning in GAMS.

**Note**

While it is not allowed to use reserved words as identifiers and labels in general, it is still possible (put not recommended) in certain cases which are explained in more detail further below.

- abort
- acronym[s]
- alias
- all
- and
- binary
- break
- card
- continue
- diag
• display
• do*
• else
• elseif
• endfor*
• endif*
• endloop*
• endwhile*
• eps
• equation[s]
• execute
• execute_load
• execute_loaddc
• execute_loadhandle
• execute_loadpoint
• execute_unload
• execute_unloaddi
• execute_unloadidx
• file[s]
• for
• free
• function[s]
• if
• inf
• integer
• logic
• loop
• model[s]
• na
• negative
• nonnegative
• no
• not
• option[s]
• or
• ord
• parameter[s]
• positive
• procedure[s] (deprecated)
• prod
• put
• put_utility/put_utilities
• putclear
• putclose
• putfmcl
• puthd
• putheader
• putpage
• puttitle
• puttl
• repeat
• sameas
• sand
• scalar[s]
• semicont
• semiint
• set[s]
• singleton
• smax
• smin
• solve
• sor
• sos1
• sos2
• sum
• system
• table[s]
• then*
• undf
• until
• variable[s]
• while
• xor
• yes
Attention

Some of the keywords above can actually be used as an identifier (e.g. `sameas`). But if they get used as identifier, their built-in meaning as part of the GAMS language can not be accessed anymore.

Note

The words marked with * in the list above are no reserved words by default. However, they get a special meaning if the dollar control option `$onEnd` is set.

The following list shows words which have a special meaning in GAMS (e.g. they are part of the solve statement), but can be used as identifiers anyway:

- eq
- eqv
- ge
- gt
- imp
- le
- lt
- maximizing
- minimizing
- ne
- scenario
- using

The reserved non-alphanumeric symbols are:

- ..
- =l=
- =g=
- =e=
- =n=
- =x=
- =c=
- =b=
- --
- ++
- **
- ->
- <=>
4.22.4.3 Identifiers

Identifiers are the names given to sets, parameters, variables, models, etc. GAMS requires an identifier to start with a letter followed by more letters or digits. The length of an identifier is currently limited to 63 characters. Identifiers may only contain alphanumeric characters (letters or numbers) or underscores (_). Examples of legal identifiers are:

a   a15   revenue   x0051

Note that the following identifiers are incorrect:

15   $casg   milk&meat

Attention

A name used for one data type cannot be reused for another.

4.22.4.4 Labels

Labels are set elements. They may be up to 63 characters long and may be used in quoted or unquoted form.

The unquoted form is simpler to use but places restrictions on characters allowed, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, underscores (_) or the sign characters + and -. Examples of valid unquoted labels are:

Phos-Acid   1986   1952-53   A
September   H2SO4   Line-1

Quotes can be used to delimit labels. Quoted labels may begin with and include any legal character. Either single or double quotes may be used but the closing quote has to match the opening quote. A label quoted with double quotes may contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances though. If we want to make a label stand out, we could put asterisks in it and indent it. A more subtle example is that GAMS keywords may be used as labels if they are quoted. So labels like parameter, put or while may be used if they are quoted. Some examples of quoted labels follow:

'   *TOTAL*   'MATCH'   '10%-INCR'   '12' / FOOT'   "line 1"

Note

- Labels do not have any numerical value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'. One can access the numerical value of a label with the Set Attributes .val attribute.
- Leading blanks in a label are significant and preserved while trailing blanks are trimmed. So 'label1' is different from 'label1', but 'label2' is identical to 'label2'.

To summarize, set names are identifiers and set elements are labels. An overview of the rules for constructing identifiers and labels is given in the following table.
Identifiers and set elements may also be associated with a line of descriptive text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the identifier.

Text may be quoted or unquoted. Quoted text may contain any character except the quote character used. Single or double quotes may be used but they must match. Text has to fit on one line and cannot exceed 255 characters in length. Text used in unquoted form must follow a number of mild restrictions. Unquoted text cannot start with a reserved word, ‘..’ or ‘=’ and must not include semicolons ‘;’ , commas ‘,’ or slashes ‘/’. End of lines terminate a text. These restrictions are a direct consequence of the GAMS syntax and are usually followed naturally by the user. Some examples are:

```
this is text
final product shipment (tpy)
"quoted text containing otherwise illegal characters ; /,"
'use single quotes to put a "double" quote in text'
```

4.22.4.6 Numbers

Numeric values are entered in a style similar to that used in other computer languages

Attention

- Blanks cannot be used in a number: GAMS treats a blank as a separator.
- The common distinction between real and integer data types does not exist in GAMS. If a number is used without a decimal point it is still stored as a real number.

In addition, GAMS uses an extended range arithmetic that contains special symbols for infinity (INF), negative infinity (-INF), undefined (UNDF), epsilon (EPS), and not available (NA). The user cannot enter UNDF; it is only produced by an operation that does not have a proper result, such as division by zero. All the other special symbols may be entered and used as if they were ordinary numbers. For more details, see section Extended Range Arithmetic.

The following example shows various legal ways of entering numbers:

```
0       156.70    -135    .095     1.
2e10    2e+10     15.e+10  .314e5   +1.7
0.0     .0        0.      INF      -INF
EPS     NA
```

The letter e denotes the well-known scientific notation allowing convenient representation of very large or small numbers. For example,

```
1e-5 = 1 * 10^{-(5)} = 0.00001;
3.56e6 = 3.56 * 10^6 = 3,560,000;
```
Note

- GAMS uses a smaller range of numbers than many computers are able to handle. This has been done to ensure that GAMS programs will behave in the same way on a wide variety of machines, including personal computers. GAMS will create an error if a number with an absolute value greater or equal to \(1.0e+300\) is used.

- A number may be entered with up to 16 significant digits. The dollar control option \$offDigit can be used to control the behavior if this number is exceeded.

4.22.4.7 Delimiters

As mentioned before, statements are separated by a semicolon ‘;’. However, if the next statement begins with a reserved word (often called keyword in succeeding chapters), then GAMS does not require that the semicolon is used.

The characters comma ‘,’ and slash ‘/’ are used as delimiters in data lists, to be introduced later. The comma terminates a data element (as does an end-of-line) and the slash terminates a data list.

4.22.4.8 Comments

A comment is an explanatory text that is not processed or retained by the computer. There are several ways to include comments in a GAMS program.

4.22.4.8.1 Blank Lines  The user may freely enter blank lines to set off certain sections and enhance readability. For example, in transport there are blank lines between the different parameters:

```plaintext
Sets
   i "canning plants" / seattle, san-diego /
   j "markets" / new-york, chicago, topeka /;

Parameters
   a(i) "capacity of plant i in cases"
       / seattle 350
       san-diego 600 /
   b(j) "demand at market j in cases"
       / new-york 325
       chicago 300
       topeka 275 /;
```

4.22.4.8.2 Single Line Comments  Users may insert a single line comment on any line by placing an asterisk ‘*’ in column 1. The text that follows the asterisk is the comment and may contain GAMS reserved words, messages or any other content. It is completely ignored by the GAMS compiler. Note that several successive lines may be single line comments as in the example below.

The default asterisk ‘*’ may be replaced by other symbols. GAMS provides the dollar control option \$comment to customize the comment character (for more on dollar control options, see chapter Dollar Control Options). The new comment character cannot be used in column 1 as before, since now it has a special meaning. The change of comment character should be used with great care. An example can be seen here:
*normal comment
*next line is a deactivated GAMS statement
* x=sum(I,z(i));
$comment !
!comment with new character
$comment *
*now we are back to how it should be

In the fourth line, the new comment character ! replaces the GAMS default * as the comment delimiter. Note that single line comments appear in the echo print of the GAMS output as numbered lines. For details see section The Echo Print of the Input File.

4.22.4.8.3 Block Comments For longer comments special 'block' delimiters may be used that cause GAMS to ignore an entire section of the program. The dollar control option $ontext marks the beginning of the comment block and the option $offtext marks the end. Note that the $ symbol must be in the first character position. The example below illustrates the use of the block comment and also contains some useful information. For more on dollar control options, see chapter Dollar Control Options.

$ontext
Following a $ontext directive in column 1 all lines are ignored by GAMS but printed on the output file until the matching $offtext is encountered, also in column 1. This facility is often used to logically remove parts of programs that are not used every time, such as statements producing voluminous reports. Every $ontext must have a matching $offtext in the same file
$offtext

Note that block comments appear in the echo print without line numbers. For details see section The Echo Print of the Input File.

4.22.4.8.4 End-of-Line Comments Comments may also be placed at the end of a line that contains GAMS code. The dollar control option $onEolCom activates end-of-line comments. The default symbol to indicate that the comment begins is a double exclamation mark !!. This symbol may be reset with the option $eolCom followed by the desired symbol which may be one character or a two-character sequence. The following example serves as illustration. For more on dollar control options, see chapter Dollar Control Options.

Scalar x /0/;
$onEolCom
x=x+1;  !! eol comment
x = x  !! eol comment in line of GAMS statement, where the GAMS statement continues to the next line +1;
$eolCom &&
x=x+1;  && eol comment with new symbol

Note that the option to add end-of-line comments may be deactivated with the dollar control option $offEolCom. End-of-line comments appear in the echo print on the appropriate lines. For details see section The Echo Print of the Input File.
4.22.4.8.5  In-Line Comments Comments may also appear in a line of GAMS code. The dollar control option $onInLine activates in-line comments. By default, the in-line comment symbols are set to the two character pairs '/*' and '*/', where '/*' indicates that the in-line comment begins and '*/' indicates that the in-line comment ends. The comment may span lines till the end-of-comment characters are encountered. The in-line comment symbols may be reset with the option $inLineCom followed by the desired pair of characters. The following example serves as illustration. For more on dollar control options, see chapter Dollar Control Options.

Scalar x /0/;
$onInLine
x=x /* in-line comment*/ +1;
$offInLine
x=x /* in-line comment in line that continues to next line */ +1;
$onLineCom /& &/
x=x /& in-line comment with new character &/ +1;

Note that the option to add in-line comments may be deactivated with the dollar control option $offInLine. In-line comments appear in the echo print on the appropriate lines. For details see section The Echo Print of the Input File. Note that in-line comments may be allowed to be nested using the dollar control option $onNestCom.

4.22.4.8.6  Outside Margin Comments GAMS provides the facility to define margins. The active code is within the margins, everything outside the set margins is ignored by the compiler and treated as comment. The dollar control option $onMargin activates margin marking and $offMargin deactivates it. The option $minCol is used to specify the first column, where GAMS code that is to be compiled may appear. Similarly, the option $maxCol is used to specify the last column for GAMS code. The following example shows how this works. For more on dollar control options, see chapter Dollar Control Options.

$context

1 2 3 4 5 6
12345678901234567890123456789012345678901234567890
$offtext

$onMargin minCol 20 maxCol 45
Now I have turned on the margin marking. Set i plant /US, UK/
Scalar x / 3.145 / Define some parameters.
Parameter a, b;

$offMargin

The text before column 20 and after column 45 is treated as a comment. Note that the full content of the lines is copied to the echo print, including everything outside the margins. For details see section The Echo Print of the Input File.

4.22.4.8.7  Hidden Comments Finally, GAMS also allows hidden comments, that are not copied to the echo print of the GAMS output file. These comments contain information that is only relevant for the person manipulating the file. They are single line comments starting with the dollar control option $hidden. An example follows. For more on dollar control options, see chapter Dollar Control Options.

$hidden a comment I do not want in LST file
set a /a1,a2/;
set b /a2,c2/;
set c /a3,d3/;
This completes the discussion of the components of the GAMS language. Many unfamiliar terms used in this chapter are further explained in the Glossary.

4.23 Set Definition

4.23.1 Introduction

Sets are fundamental building blocks in any GAMS model. They allow the model to be succinctly stated and easily read. In this chapter we will introduce how sets are declared and initialized. More advanced set concepts, such as assignments to sets, and lag and lead operations are covered in the chapters Dynamic Sets and Sets as Sequences. The topics discussed in this chapter will be enough to provide a good start on most models. We will introduce simple sets, subsets, multi-dimensional sets, singleton sets and the universal set. The chapter will be concluded by a topic on domain checking, a very important feature of GAMS, and a section about Domain Defining Symbol Declarations.

4.23.2 Simple Sets

Using common mathematical notation, a set \( S \) that contains the elements \( a \), \( b \) and \( c \) is written as:

\[
S = \{a, b, c\}
\]

Using GAMS notation, the same set is defined in the following way:

```gams
set S / a, b, c /;
```

The set statement begins with the keyword `set`, \( S \) is the name of the set, and its members are \( a \), \( b \), and \( c \). They are labels, but are often referred to as elements or members.

4.23.2.1 Defining a Simple Set: The Syntax

In general, the syntax for simple sets in GAMS is as follows:

```gams
set [s] set_name ["text"] [/element [text] {,element [text]} /]
   {,set_name ["text"] [/element [text] {,element [text]} /] } ;
```

`Set [s]` is the keyword that indicates that this is a set statement. `Set_name` is the internal name of the set in GAMS, it is an identifier. The optional explanatory text may be used to describe the set or a specific set element for future reference and to ease readability. The list of set elements is delimited by forward slashes. `Element` is the name of the set element(s). Note that each element in a set must be separated from other elements by a comma or by an end-of-line, and each element is separated from any associated text by a blank.

Consider the following example from the Egyptian fertilizer model [FERTS], where the set of fertilizer nutrients could be written as
The order in which the set members are listed is usually not important. However, if the members represent, for example, time periods, then it may be useful to refer to the next or previous member. There are special operations to do this, and they are discussed in chapter Sets as Sequences: Ordered Sets. For now, it is enough to remember that the order in which set elements are specified is not relevant, unless and until some operation implying order is used. At that time, the rules change, and the set becomes what we will later call an ordered set.

Note

- The data statement, i.e. specification of set elements in forward slashes can be omitted. In such cases a set is declared without being defined.
- More than one set may be declared and defined in one set statement. Examples are given in subsection Declaring Multiple Sets below.

### 4.23.2.2 Illustrative Examples

Consider the following example based on the model [SHALE]:

```plaintext
Set cf "final products" / syncrude "refined crude (mil bbls)"
     lpg  "liquefied petroleum gas (million bbls)"
     ammonia "ammonia (mil tons)"
     coke  "coke (mil tons)"
     sulfur "sulfur (mil tons)" /;
```

The set statement is introduced with the keyword `set`, the name of the set is `cf` and the explanatory text "final products" describes the set. The set has five elements with explanatory texts that contain details of the units of measurement.

Usually sets are declared and defined once and then referenced in the model. There are two exceptions: the dollar control option `onMulti` allows adding more elements later, and dynamic sets. For details on dynamic sets, see chapter Dynamic Sets. The following code sightly varies the previous example to demonstrate the option `$onMulti`:

```plaintext
Set cf "final products" / syncrude "refined crude (mil bbls)"
     lpg  "liquefied petroleum gas (million bbls)"
     ammonia "ammonia (mil tons)" /
$onmulti
Set cf "more final products" / coke  "coke (mil tons)"
     sulfur "sulfur (mil tons)" /;
```

After `$onmulti` additional elements are added to the set `cf`. Note that without the dollar control option `$onMulti` that would generate an error as per default a symbol can have at most one data statement.
4.23 Set Definition

4.23.2.3 Sequences as Set Elements

The asterisk ‘∗’ plays a special role in set definitions. It is used to relieve the tedium of typing a sequence of elements for a set, and to make intent clearer. For example, in a simulation model there might be ten annual time periods from 1991 to 2000. Instead of typing ten years, the elements of this set can be written as:

Set t "time" / 1991 * 2000 /;

This means that the set includes the ten elements 1991, 1992, ..., 2000. GAMS builds up these label lists by looking at the differences between the two labels. If the only characters that differ are digits, with the number L formed by these digits in the left and R in the right, then a label is constructed for every integer in the sequence L to R. Any non-numeric differences or other inconsistencies cause errors.

The following example illustrates the most general form of the 'asterisked' definition:

Set g1 / a1bc * a20bc /;

Note that this is not the same as:

Set g2 / a01bc * a20bc /;

Both sets have 20 members, but they have only 11 members in common.

Lists in decreasing order are also possible:

Set y "years in decreasing order" / 2000 * 1991 /;

As a last example, the following set definitions are both illegal because they are not consistent with the rule given above for making lists:

Set illegal1 / a1x1 * a9x9 /;
Set illegal2 / a1 * b9 /;

4.23.2.4 Declaring Multiple Sets

The keyword set does not need to be used for each set, rather only at the beginning of a group of sets. It is often convenient to put a group of set declarations (and definitions) together at the beginning of the program. When this is done the set keyword needs only be used once. Those who prefer to intermingle set declarations with other statements, have to use a new set statement for each additional group of sets. Note that the keywords set and sets are equivalent. The following example below shows how two sets can be declared together. Note that the semicolon is used only after the last set is declared.

Sets s "Sectors" / manuf, agri, services, government /;
Sets r "Regions" / north, eastcoast, midwest, sunbelt /;
4.23.2.5 Using Previously Defined Sets in Set Definitions

The following notation allows previously defined sets to be used in a new set definition:

Set i / i1 * i4 /
   j / j6 * j9 /
   k / #i, set.j /;

The set k contains all elements of the sets i and j. Note that the hash sign '#' followed by a set name is a shorthand for referring to all the elements in a set. The notation set.set.name works identically and is just a different way to refer to all elements in a previously defined set.

4.23.3 The Alias Statement: Multiple Names for a Set

Sometimes it is necessary to have more than one name for the same set. In input-output models for example, each commodity may be used in the production of all other commodities and it is necessary to have two names for the set of commodities to specify the problem without ambiguity. In the general equilibrium model [ORANI], the set of commodities c is written as

Set c "Commodities" / food, clothing /;

A second name for the set c is established with either of the following statements:

Alias (c, cp) ;
Alias (cp, c) ;

Here cp is the new set name that can be used instead of the original set name c.

Note
The newly introduced set name may be used as an alternative name for the original set; the associated set will always contain the same elements as the original set.

With the alias statement more than one new name may be introduced for the original set:

Alias (c,cp, cpp, cppp);

Here cp, cpp, ccppp are all new names for the original set c.
4.23 Set Definition

Note

The order of the set names in the alias statement does not matter. The only restriction is that exactly one of the sets in the statement must be defined earlier. All the other sets are introduced by the alias statement.

Typical examples for the usage of aliases are problems where transportation costs between members of one set have to be modeled. The following code snippet is adapted from the Andean fertilizer model [ANDEAN]:

```gams
Set i "plant locations" / palmasola, pto-suarez, potosi, baranquill, cartagena /;
Alias(i,ip);
Table tran(i,i) "transport cost for interplant shipments (us$ per ton)"
     palmasola pto-suarez potosi baranquill
pto-suarez 87.22
potosi   31.25 55.97
baranquill 89.80 114.56 70.68
cartagena 89.80 114.56 70.68  5.00
;
Parameter mui(i,ip) "transport cost: interplant shipments (us$ per ton)";
mui(i,ip) = (tran(i,ip) + tran(ip,i));
```

The alias statement introduces `ip` as another name for the set `i`. The `table tran` is two-dimensional and both indices are the set `i`. The data for the transport cost between the plants is given in this table; note that the transport costs are given only for one direction here, i.e. the costs from `pto-suarez` to `palmasola` are explicitly specified in the table while the costs in the opposite direction are not given at all. The parameter `mui` is also two-dimensional and both indices refer to the set `i`, but this time the alias `ip` is used in the second position. The parameter `mui` is defined with the assignment statement in the next line: `mui` contains the transport costs from one plant location to the other, in both directions. Note that if `mui` were defined without the alias, then all its entries would have been zero. For other examples where aliases are used, see sections The Universal Set and Finding Sets from Data below.

4.23.4 Subsets

It is often necessary to define sets whose members must all be members of some larger set. The syntax is:

```gams
set set_ident1(set_ident2) ;
```

Here `set` is the keyword indicating that this is a set statement, and `set_ident1` is a subset of the larger set `set_ident2`. The larger set is also called superset.

For instance, we may wish to define the sectors in an economic model following the style in [CHENERY].

```gams
Set i "all sectors" / light-ind, food+agr, heavy-ind, services /
t(i) "traded sectors" / light-ind, food+agr, heavy-ind /
nt "non-traded sectors" / services /;
```

Some types of economic activity, for example exporting and importing, may be logically restricted to a subset of all sectors. In order to model the trade balance we need to know which sectors are traded, and one obvious way is to list them explicitly, as in the definition of the set `t` above. The specification `t(i)` means that each member of the set `t` must also be a member of the set `i`. GAMS will enforce this relationship, which is called domain checking. Obviously, the order of declaration and definition is important: the membership of `i` must be known before `t` is defined, otherwise checking cannot be done.
Note

All elements of the subset must also be elements of the superset.

It is legal but unwise to define a subset without reference to the larger set, as is done above for the set nt. In this case domain checking cannot be performed: if services were misspelled no error would be marked, but the model may give incorrect results. Hence, it is recommended to use domain checking whenever possible. It catches errors and allows to write models that are conceptually cleaner because logical relationships are made explicit.

An alternative way to define elements of a subset is with assignments:

Set i "all sectors" / light-ind, food+agr, heavy-ind, services /
   t(i) "traded sectors" / light-ind, heavy-ind /;
t('food+agr') = yes;

In the last line the element food+agr of the set i is assigned to the subset t. Assignments may also be used to remove an element from a subset:

t('light-ind') = no;

Note that yes and no are reserved words in GAMS. Note further that if a subset is assigned to, it then becomes a dynamic set. For more on assignments in GAMS in general, see section The Assignment Statement.

Attention

A subset can be used as a domain in the declaration of other sets, variables, parameters and in equations as long as it is no dynamic set.

This completes the discussion of sets in which the elements are simple. This is sufficient for many GAMS applications. However, there are a variety of problems for which it is useful to have sets that are defined in terms of two or more other sets.

4.23.5 Multi-Dimensional Sets

It is often necessary to provide mappings between elements of different sets. For this purpose, GAMS allows the use of multi-dimensional sets. For the current maximum number of permitted dimensions, see Dimensions. The next two subsections explain how to express one-to-one and many-to-many mappings between sets.

4.23.5.1 One-to-one Mapping

Consider a set whose elements are pairs: \( A = \{ (b, d), (a, c), (c, e) \} \). In this set there are three elements and each element consists of a pair of letters. This kind of set is useful in many types of modeling. For example, in the world aluminum model \([\text{ALUM}]\) a port has to be associated with a nearby mining region

Set i "mining regions" / china, ghana, ee+ussr, s-leone /
n "ports" / accra, freetown, leningrad, shanghai /
in(i,n) "mines to ports map" / china .shanghai
   ghana .accra
   ee+ussr.leningrad
   s-leone.freetown /;

Here i is the set of mining regions, n is the set of ports and in is a two dimensional set that associates each port with a mining region. The dot between china and shanghai is used to create one such pair. Blanks may be used freely around the dot for readability. The set in has four elements, and each element consists of a region-port pair. The notation \((i,n)\) after the set name in indicates that the first member of each pair must be a member of the set i of mining regions, and that the second must be in the set n of ports. GAMS will domain check the set elements to ensure that all members belong to the appropriate sets.
4.23 Set Definition

4.23.5.2 Many-to-Many Mapping

A many-to-many mapping is needed in certain cases. Consider the following sets:

Set  i / a, b / 
    j / c, d, e / 
    ij1(i,j) / a..c, a..d / 
    ij2(i,j) / a..c, b..c / 
    ij3(i,j) / a..c, b..c, a..d, b..d /;

Here the set \( ij1 \) presents a one-to-many mapping where one element of the set \( i \) maps onto many elements of the set \( j \). The set \( ij2 \) represents a many-to-one mapping where many elements of the set \( i \) map onto one element of the set \( j \). The set \( ij3 \) is the most general case: a many-to-many mapping where many elements of the set \( i \) map to many elements of the set \( j \).

These sets may be written compactly as:

Set  i / a, b / 
    j / c, d, e / 
    ij1(i,j) / a.(c,d) / 
    ij2(i,j) / (a,b).c / 
    ij3(i,j) / (a,b).(c,d) /;

The parenthesis provides a list of elements that is expanded when creating pairs. Note that the dot ‘.’, if used like above, acts as product operator and supports building the Cartesian product of sets.

Attention

When complex sets like this are created, it is important to check that the desired set has been obtained. The checking can for example be done be done by using a display statement.

GAMS provides more notation to define multi-dimensional sets in a succinct way. As introduced above the hash sign ‘#’ followed by a set name is a shorthand for referring to all the elements in a set. The matching operator ‘::’ may be used to map ordered sets. This operator is similar to the product operator ‘.’. However, in this case elements are matched pairwise by mapping elements with the same order number. The examples below demonstrate these concepts.

Set  i / a, b / 
    j / c, d, e / 
    ij4a(i,j) / a.#j / 
    ij4b(i,j) / a.c, a.d, a.e / 
    ij5a(i,j) / #i.#j / 
    ij5b(i,j) / a.c, a.d, a.e, b..c, b..d, b..e / 
    ij6a(i,j) / #i:#j / 
    ij6b(i,j) / a.c, b..d /;

Note that set names that differ only by the last letter denote identical sets. For example, set \( ij4a \) is identical to set \( ij4b \). Observe that set \( i \) has two elements and set \( j \) has three elements, where \( e \) is the element with the highest order. Set \( ij6a \) is an ordered mapping of all elements of set \( i \) to all elements of set \( j \). However, since there is a mismatch in the number of elements, element \( e \) is not mapped to.

These concepts may be generalized to sets with higher dimensions. Mathematically, these are called 3-tuples, 4-tuples, or more generally, \( n \)-tuples. Some examples for the compact representation of sets of \( n \)-tuples using combinations of dots, parentheses, and commas are shown in Table 1.
A powerful and very compact way to define multi-dimensional sets is with a special option that takes an identifier as value and carries out identifier operations like index matching using the matching operator ':'. The following example illustrates the method.

```plaintext
Set i / i1*i4 /
j / j1*j5 /
k / k1,k2 /
h / h1*h3 /;
Set b(i,j,k), c(i,j,k,h);
Option b(i:j,k), c(b:h);
display b, c;
```

The set \( b \) is a three-dimensional set, the option statement specifies which permutations of the elements of \( i, j, \) and \( k \) are elements of \( b \). The matching operator ': ' is between \( i \) and \( j \), so we must first match the elements of the sets \( i \) and \( j \). That gives us the the first two positions. For the third position we cycle through all elements of the set \( k \). This results in the following elements for the set \( b \):

\[
i1.j1.k1, i1.j1,k2, i2.j2.k1, \ldots , i4.j4.k2
\]

The set \( c \) is a four-dimensional set. Note that the first three dimensions are identical to the domain of the set \( b \). The option statement specifies that in the first three positions we will have elements of the set \( b \) and and these are matched with the elements of the set \( h \) which are in the fourth position. Now, the set \( h \) has only three elements, so only the first three elements of the set \( b \) are matched with the members of the set \( h \). This results in the following set:

\[
i1.j1.k1.h1, i1.j1.k2.h2, i2.j2.k1.h3
\]

As recommended above, it is important to always check whether the multi-dimensional sets generated with compact statement like these are indeed the sets that were intended.

For more sophisticated examples of how to use the matching operator within an option statement please see section Index Matching .
### 4.23.5.3 The Table Format for Multi-Dimensional Sets

An alternative way to declare multi-dimensional sets is with tables. We show by example how tables may be used in the context of set definitions:

Set origins / Berlin, Paris /
  destinations / London, Chicago, Budapest /
  linked_1(origins,destinations) "cities linked by railways"
  / Berlin.London, Berlin.Budapest, 

Set Table linked_2(origins,destinations)

<table>
<thead>
<tr>
<th></th>
<th>London</th>
<th>Chicago</th>
<th>Budapest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Paris</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The set linked_1 is a two-dimensional set that is defined with the dot notation introduced above. The set linked_2 is the same set defined using the table notation: the keyword set is followed by the keyword table and the name of the set with its domain. The table itself consists of the elements of the first index in the first column, the elements of the second index in the first row, and the data in the grid positions. Note that the keyword yes indicates that a label combination is part of the two-dimensional set and the keyword no or a blank indicates that the label combination is not contained in the new set. Please see section Tables for detailed requirements for inputting data in the table format.

Alternatively, the multi-dimensional set may be declared first without any elements, and the elements are added later in a separate table statement:

Set origins / Berlin, Paris /
  destinations / London, Chicago, Budapest /
  linked_2(origins,destinations) "cities linked by railways";

Table linked_2(origins,destinations)

<table>
<thead>
<tr>
<th></th>
<th>London</th>
<th>Chicago</th>
<th>Budapest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Paris</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instead of the keywords yes and no users may also use numbers to specify membership in the two-dimensional set: nonzero numeric entries mean that a label combination is part of the set and zero or a blank indicates that the label combination is not contained in the set.

### 4.23.5.4 Projection and Aggregation of Sets

In GAMS, projection and aggregation operations on sets can be performed in two different ways: with an option statement and with an assignment. For a detailed discussion, see section Projection and Aggregation of Sets and Parameters.
4.23.6 Singleton Sets

A singleton set in GAMS is a special set that has at most one element (zero elements are allowed as well). Like other sets, singleton sets may have a domain with several dimensions. For the current maximum number of permitted dimensions, see Dimensions. Singleton sets are declared and defined with the keyword singleton that acts as a modifier to the keyword set:

```
Set i / a, b, c /
Singleton Set j / d /
   k(i) / b /
   l(i,i) / b.c /
```

The sets j, k and l are declared as singleton sets, each of them has just one element. The set k is a subset of the set i and the set l is a two-dimensional set.

Note that a data statement for a singleton set with more than one element will create a compilation error:

```
1 Singleton Set s / s1*s3 /;
 **** $844
2 display s;
```

Error Messages

844 Singleton with more than one entry (see $onStrictSingleton)

It also possible to assign an element to a singleton set. In this case the singleton set is automatically cleared of the previous element first. For example, adding the following line to the code above will result in set k containing only element a after execution:

```
k('a') = yes;
```

The dollar control option offStrictSingleton may be used to allow sets that are declared as singleton sets to have more than one element in compile time definitions. However, in this case only the first listed element is a valid element of the set. Note that the value of zero for the command line parameter strictSingleton has the same effect for execution time definitions of singleton sets via assignment statements.

For more on dollar control options, see chapter Dollar Control Options. For more on GAMS command line parameters, see chapter The GAMS Call and Command Line Parameters. For more on compilation errors, see section Compilation Errors.

Singleton sets can be especially useful in assignment statements since they do not need to be controlled by a controlling index or an indexed operator like other sets. Consider the following examples:

```
Set i / a, b, c /
Singleton Set k(i) / b /
   h(i) / a /
Parameter n(i) / a 2, b 3, c 5 /
Scalar z1, z2;

z1 = n(k);
z2 = n(k) + 100*n(h);
```

The singleton sets k and h are both subsets of the set i. The parameter n is defined over the set i. The scalar z1 is assigned a value of the parameter n without naming the respective label explicitly in the assignment. It is already specified in the definition of the singleton set k. The assignment statement for the scalar z2 contains an expression where the singleton sets k and h are referenced without a controlling index or an indexed operation.

Note

Singleton sets cannot be used as domains.
4.23.7 The Universal Set: * as Set Identifier

GAMS provides the universal set denoted by '*' for cases where the user wishes not to specify an index but have only a placeholder for it. The following examples show two ways how the universal set is introduced in a model. We will discuss the advantages and disadvantages of using the universal set later. The first example is from the production and inventory model [ROBERT]:

\begin{verbatim}
Sets r "raw materials" / scrap, new /
Table misc(*,r) "other data"
   scrap new
max-stock   400  275
storage-c   .5   2
res-value   15   25;
\end{verbatim}

A table is an input format for the data type parameter and has at least two dimensions. For details see section Tables. In our example, the first index is the universal set '*' and the second index is the previously defined set r. Since the first index is the universal set any entry whatsoever is allowed in this position. In the second position elements of the set r must appear, they are domain checked, as usual.

The second example illustrates how the universal set is introduced in a model with an alias statement.

\begin{verbatim}
Alias (new_universe,*);
Set k(new_universe) / Chicago /;
\end{verbatim}

The alias statement links the universal set with the set name new_universe. Set k is a subset of the universal set and Chicago is declared to be an element of k. Any item may be added freely to k.

The universal set is particularly useful for generating reports, since it allows the use of any labels without having to define special sets for them. For an example, see section Set Attributes below. For more on report writing, see chapter The Put Writing Facility.

Attention

It is recommended not to use the universal set for data input, since there is no domain checking and thus typos will not be detected and data that the user intends to be in the model might actually not be part of it.

Observe that in GAMS a simple set is always regarded as a subset of the universal set. Thus the set definition

\begin{verbatim}
Set i / i1*i10 /;
\end{verbatim}

is the same as

\begin{verbatim}
Set i(*) / i1*i10 /;
\end{verbatim}

GAMS follows the concept of a domain tree for domains in GAMS. It is assumed that a set and its subset are connected by an arc where the two sets are nodes. Now consider the following one dimensional subsets:

\begin{verbatim}
Set i, ii(i), j(i), jj(j), jjj(jj);
\end{verbatim}

These subsets are connected with arcs to the set i and thus form a domain tree that is rooted in the universe node '*'. This particular domain tree may be represented as follows:

\begin{verbatim}
* - i - ii
   |    
   - j - jj - jjj
\end{verbatim}

Note that with the construct i(jjj) we may access ii iterating through the members of jjj. For an example, see domain tree in the loop statement.

Observe that the universal set is assumed to be ordered and operators for ordered sets such ord, lag and lead may be applied to any sets aliased with the universal set.
4.23.8 Set and Set Element Referencing

Sets or set elements are referenced in many contexts, including assignments, calculations, equation definitions and loops. Usually GAMS statements refer to the whole set or a single set element. In addition, GAMS provides several ways to refer to more than one, but not all elements of a set. In the following subsections we will show by example how this is done. GAMS also has set functions that specifically reference sets and are introduced in the chapter about logical conditions.

4.23.8.1 Referencing the Whole Set

Most commonly whole sets are referenced as in the following examples:

Set i / i1*i100 /;
Parameter k(i);
k(i) = 4;
Scalar z;
z = sum(i, k(i));

The parameter \(k\) is declared over the set \(i\), in the assignment statement in the next line all elements of the set \(i\) are assigned the value 4. The scalar \(z\) is defined to be the sum of all values of the parameter \(k(i)\).

4.23.8.2 Referencing a Single Element

Sometimes it is necessary to refer to specific set elements. This is done by using single or double quotes around the label(s). We may add the following line to the example above:

\[k('i77') = 15;\]

This statement changes the value of \(k('i77')\) to 15, all the other values of \(k\) remain 4.

4.23.8.3 Referencing a Part of a Set

There are multiple ways to restrict the domain to more than one element, e.g. subsets, conditionals and tuples. Suppose we want the parameter \(k\) from the example above to be assigned the value 10 for the first 8 elements of the set \(i\). The following two lines of code illustrate how easily this may be accomplished with a subset:

Set j(i) / i1*i18 /;
k(j) = 10;

First we define the set \(j\) to be a subset of the set \(i\) with exactly the elements we are interested in. Then we assign the new value to the elements of this subset. The other values of the parameter \(k\) remain unchanged. For examples using conditionals and tuples, see sections Restricting the Domain: Conditionals and Restricting the Domain: Tuples respectively.

4.23.9 Set Attributes

A GAMS set element has several numbers attached to it. These values are called attributes; they may be recovered during execution. The attributes are listed in Table 3.
### 4.23 Set Definition

<table>
<thead>
<tr>
<th>Set Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>.pos</td>
<td>Element position in the current set (set does not have to be ordered), starting with 1.</td>
</tr>
<tr>
<td>Ord</td>
<td>.ord</td>
<td>Same as .pos but for ordered sets only.</td>
</tr>
<tr>
<td>Offset</td>
<td>.off</td>
<td>Element position in the current set minus 1. So .off = .pos - 1 (set does not have to be ordered).</td>
</tr>
<tr>
<td>Reverse</td>
<td>.rev</td>
<td>Reverse element position in the current set, so the value for the last element is 0, the value for the penultimate is 1, etc. (set does not have to be ordered).</td>
</tr>
<tr>
<td>Unique Element List</td>
<td>.uel</td>
<td>Element position in the unique element list. For details see section Ordered and Unordered Sets</td>
</tr>
<tr>
<td>Length</td>
<td>.len</td>
<td>Length of the set element name (a count of the number of characters).</td>
</tr>
<tr>
<td>Value</td>
<td>.val</td>
<td>If a set element is a number, this attribute gives the value of the number. For extended range arithmetic symbols, the symbols are reproduced. If a set element is a string that is not a number, then this attribute is not defined and trying to use it results in an error.</td>
</tr>
<tr>
<td>First set element</td>
<td>.first</td>
<td>Returns 1 for the first set element, otherwise 0.</td>
</tr>
<tr>
<td>Last set element</td>
<td>.last</td>
<td>Returns 1 for the last set element, otherwise 0.</td>
</tr>
</tbody>
</table>

**Table 3: Set Attributes**

The attributes may be accessed with an assignment statement:

```plaintext
data(set_name) = set_name.attribute ;
```

Here `data` is a parameter, `set_name` is the name of the set and `.attribute` is one of the attributes listed above. The following example serves as illustration:

```plaintext
Set id "example set" / Madison, tea-time, '-inf', '-7', '13.14'/;
Parameter report(id,*) "set attribute values";

report(id,'position') = id.pos ;
report(id,'reverse') = id.rev ;
report(id,'offset') = id.off ;
report(id,'length') = id.len ;
report(id,'first') = id.first;
report(id,'last') = id.last ;

display report;
```

The parameter `report` is declared to have two dimensions with the set `id` in the first position and the universal set in the second position. In the following six statements the values of `report` are defined for six entries of the universal set. Note how the flexibility of the universal set is used here to make reporting easy. The `display statement` generates the output that follows.

```plaintext
---- 11 PARAMETER report set attribute values

<table>
<thead>
<tr>
<th>position</th>
<th>reverse</th>
<th>offset</th>
<th>length</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madison</td>
<td>1.000</td>
<td>4.000</td>
<td>7.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>tea-time</td>
<td>2.000</td>
<td>3.000</td>
<td>1.000</td>
<td>8.000</td>
<td></td>
</tr>
<tr>
<td>-inf</td>
<td>3.000</td>
<td>2.000</td>
<td>2.000</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>4.000</td>
<td>1.000</td>
<td>3.000</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>13.14</td>
<td>5.000</td>
<td>4.000</td>
<td>5.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>
```
4.23.10 Finding Sets from Data

Sometimes it is desirable to find a set from the available data in order to use it later in the model. We will show by example how this may be accomplished using the alias statement, the universal set and conditionals. Suppose we have only the data related to the transportation model [TRNSPORT] and we want to identify the sets. We can tell from the data that there are two sets that we are interested in. First, we define these two sets as aliases of the universal set, which means that no elements are specified:

Alias(sources, places, *);

Then we enter the data that contain an indicator of which set elements are valid entries in the sets to be computed. We use the table format.

<table>
<thead>
<tr>
<th>Table trandata (sources,places) &quot;data from spreadsheet&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newyork</td>
</tr>
<tr>
<td>Seattle</td>
</tr>
<tr>
<td>Sandiego</td>
</tr>
<tr>
<td>totalneed</td>
</tr>
</tbody>
</table>
| ;

Next we define subsets that we will need in the calculations that follow:

Set source(sources) "sources in spreadsheet data" destination(places) "destination in spreadsheet data";

Now we have everything that we need to do the calculation using the data on hand. In our case, a label qualifies as an element of the set source if it has an entry for totalsupply in the table above, and a label is an element of the set destination if it has an entry for totalneed in the table trandata:

source(sources)$(trandata(sources,"totalsupply")) = yes;
destination(places)$(trandata("totalneed", places )) = yes;

These conditional assignments define the elements of the sets source and destination. From this point on these sets may be used in the model. However, note that the resulting sets are dynamic sets. Hence they cannot be used as domains in declaration statements of other sets, parameters, variables and equations. But they may be referenced and used in equation definitions.

Such computations may for example be useful if the user gets a data table from elsewhere and needs to specify the sets. Alternatively, if the data is available in gdx format, the dollar control option load provides functionality to project sets from data contained in a GDX file.

See also section Implicit Set Definition (or: Domain Defining Symbol Declarations).
4.23.11 Domain Checking

The GAMS compiler performs a check to ensure that each label quoted as a member of a set is indeed an element of the respective set, and each element defined in a subset is in fact a member of the superset. This screening for consistency is called *domain checking*. It is done whenever a domain is referenced, be it in set, variable, parameter or equation declarations and definitions, or in assignments. The following examples serve as illustration.

Set i "all cities" / Lima, Toronto, Wuhan, Shanghai /
    as(i) "Asian cities" / Wuhan, Shanghai, Calcutta /
    am "American cities" / Lima, Toronto /;

The set `as` is declared to be a subset of the set `i`, therefore domain checking will test every label for inconsistencies. It will catch two errors: there is a typo in `Shanghai` and `Calcutta` is not a member of the set `i`, so it cannot legally be a member of a subset.

1 Set i "all cities" / Lima, Toronto, Wuhan, Shanghai /
2    as(i) "Asian cities" / Wuhan, Shanghai, Calcutta /

**** $170 $170
**** 170 Domain violation for element
3 am "American cities" / Lima, Toronto /;

The user can rectify the spelling error, and either delete `Calcutta` from the subset `as` or add it to the superset `i`. The following line will pass domain checking:

Set as(i) "Asian cities" / Wuhan, Shanghai /;

Note that `am` is not declared as a subset of the set `i` even though it apparently should contain cities contained in `i`. Hence, `am` cannot be domain checked and the typo in `Toronto` will go undetected. This has consequences for the next line:

Parameter pam(am) "population in millions" / Lima 8.9, Toronto 5.6 /;

In this parameter definition the domain of the parameter `pam` is the set `am`. GAMS will report an error here, since domain checking does not recognize the label `Toronto`. `Toronto`, as specified in the definition of the set `am` above would be accepted.

A further example for domain checking concerns multi-dimensional domains where the user accidentally switches the positions of the indices:

Parameter h(as,am) / Wuhan.Lima 10, Wuhan.Toronto 12, Shanghai.Lima 7/;
Parameter d(as,am);
  d(as,am) = 5*h(am,as) + 78;

Observe that we assume that the typo in the label `Toronto` has been rectified. The parameter `h` is defined over the domain `(as,am)`. However, in the assignment statement in the last line above, it is referenced with the domain `(am,as)`. This mistake is caught by domain checking and an error is reported.

As we have seen in the definition of the set `am` above, domain checking is not compulsory. If the following statement is entered, GAMS makes no assumptions about `rho` until further information is provided.
set t years / 1988 * 1995 /;
Parameter rho discount rate ;

The modeler may later choose to domain check rho by continuing the definition with the following line:

Parameter rho(t) / 1988 0.07, 1989*1994 0.10, 1995 0.09 /;

Alternatively, the modeler may choose not to domain check the parameter rho, as is shown in the deliberately nonsensical (but legal) statement that follows:

Parameter rho / 1988.January 0.07, strategy-1.cost 44, cat.capacity 99 /;

If a parameter is not domain-checked, the only restriction is that the dimensionality must be constant. Once the number of labels per data item has been established it is frozen; to refer to the parameter differently is an error.

Note

Domain checking is automatic; it is only suppressed in two cases:
1. The index is the universal set or a set aliased to the universal set, see the examples above.
2. The dollar control option $onWarning is used. It has the effect that warnings rather than errors are reported for domain violations.

We urge modelers to use domain checking whenever possible. It catches errors and allows users to write models that are conceptually cleaner because logical relationships are made explicit.

Note that the dollar control option $load is available in several variations to enable domain checking when loading data from a GDX file. For details, see $loadDC, $loadDCM and $loadDCR and chapter GAMS Data eXchange (GDX).

4.23.12 Implicit Set Definition (or: Domain Defining Symbol Declarations)

As seen above, sets can be defined through data statements in the declaration. Alternatively, sets can be defined implicitly through data statements of other symbols which use these sets as domains. This is illustrated in the following example, which is derived from the [TRNSPORT] model:

Set
   i 'canning plants'
   j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
    new-york  chicago  topeka
     seattle    2.5   1.7    1.8
     san-diego  2.5   1.8    1.4;

Display i,j;

Note the < signs in the domain list of the declaration of d (d(i<,j<)). These signal, that the set i will contain all elements which define the first dimension of symbol d and that the set j will contain all elements which define the second dimension of symbol d, respectively. So, this is the output of the Display statement at the end:
This syntax is not limited to the table statement, but can be used with any symbol declaration. Also, one domain set can be defined through multiple symbols using the same domain, when using the dollar control option onMulti:

Set
  food
    fruits(food<) / apple, orange /
  $onMulti
    vegetable(food<) / carrot, cauliflower /
    meat(food<) / beef, pork /;

Display food;

This is the output of the Display statement:

---- 8 SET food
apple , orange , carrot , cauliflower, beef , pork

Note
If the < sign is used to mark a declaration as "domain defining", this attribute is not limited to the data statement following this declaration, but also influences other ways to define data at compile time like the dollar control option load, as shown in the following example:

Set
  i 'canning plants'
  j 'markets';

Parameter d(i<,j<) 'distance in thousands of miles';

$gdxIn data.gdx
$load d

Attention
Only non-zero elements in a symbol will add elements to an implicitly defined set. This is illustrated in the following two examples.

Set
  i 'canning plants'
  j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
  new-york    chicago    topeka
  seattle  2.5        1.8
  san-diego 2.5        1.4;

Display i,j;
Note the empty column for chicago. Since there is no data, chicago will not end up in the set j, which can be seen in the output of the Display statement:

```
---- 10 SET j markets
new-york, topeka
```

Also, an explicit 0 in a data statement does not add elements to an an implicitly defined set (in contrast to an eps). This is shown in the following GAMS code and output:

```
Set
  j 'markets';

Parameter
  b(j) 'demand at market j in cases'
    / new-york 325
    chicago 0
    topeka eps /;

Display j;
```

```
---- 10 SET j markets
new-york, topeka
```

See also section Finding Sets from Data.

4.23.13 Summary

In GAMS, a simple set consists of a set name and the elements of the set. Both the name and the elements may have associated text that explains the name or the elements in more detail. More complex sets have elements that are pairs or even n-tuples. These sets with pairs and n-tuples are ideal for establishing relationships between the elements in different sets. GAMS also uses a domain checking capability to help catch labeling inconsistencies and typographical errors made during the definition of related sets.

The discussion here has been limited to sets whose members are all specified as the set is being declared. For many models this is all the user needs to know about sets. Later we will discuss more complicated concepts, such as sets whose membership changes in different parts of the model (assignment to sets) and other set operations such as unions, complements and intersections.

4.24 Dynamic Sets

4.24.1 Introduction

Sets in general are covered in chapter Set Definition. In this chapter we introduce a special type of sets: dynamic sets. The sets that we discuss in detail in chapter Set Definition have their elements stated at compile time (e.g. enclosed in slashes at the set declaration or when loading a set from gdx via $load) and during execution time the membership is never changed. Therefore they are called static sets. In contrast, the elements of dynamic sets are not fixed, but may be added and removed during execution of the program. Dynamic sets are most often used as controlling indices in assignments or equation definitions and as the conditional set in a dollar-controlled indexed operation. We will first show how assignments are used to change set membership in dynamic sets. Then we will introduce set operations and the last part of this chapter covers dynamic sets in the context of dollar conditions.
4.24 Dynamic Sets

4.24.2 Assigning Membership to Dynamic Sets

Dynamic Sets may be assigned to in a similar way as other data types. There are only two possible values: yes and no. Note that arithmetic operations cannot be performed on sets in the same way as on value typed identifiers (parameters, variables and equations subtypes). However, there are special set operations.

4.24.2.1 The Syntax

Like any other set, a dynamic set has to be declared before it may be used in the model. Often, a dynamic set is declared as subset of a static set. Dynamic sets in GAMS may also be multi-dimensional like static sets. The maximum number of permitted dimensions follows the rules of the basic Data Types and Definitions. For multi-dimensional dynamic sets the index sets can also be specified explicitly at declaration. That way dynamic sets are domain checked. Of course it is also possible to use dynamic sets that are not domain checked. This provides additional power and flexibility but also a lack of intelligibility and danger. Any label is legal as long as such a set's dimension, once established, is preserved.

In general, the syntax for assigning membership to dynamic sets in GAMS is:

\[
\text{set_name(index_list | label) = yes | no ;}
\]

Set name is the internal name of the set in GAMS, index_list refers to the domain of the dynamic set and label is one specific element of the domain. An assignment statement may assign membership to the dynamic set either to the whole domain or to a subset of the domain or to one specific element. Note that, as usual, a label must appear in double or single quotes. Yes and no are keywords in GAMS. They are used to add members to or remove them from the dynamic set. Examples are given in the following subsections.

4.24.2.2 Illustrative Example

Throughout this chapter we will use examples adapted from the database model [ZLOOF] to illustrate the introduced concepts. Here we start with assignments of membership to dynamic sets.

```
Set  item    "all items" / dish, ink, lipstick, pen, pencil, perfume /
    subitem1(item) "first subset of item" / pen, pencil /
    subitem2(item) "second subset of item";

subitem1('ink') = yes ;
subitem1('lipstick') = yes ;
subitem2(item) = yes ;
subitem2('perfume') = no ;
display subitem1, subitem2;
```

Note that the sets subitem1 and subitem2 are declared like any other set. The two sets become dynamic as soon as they are assigned to a few lines later. They are also domain checked: the only members they will ever be able to have must also be members of the set item. The first assignment not only makes the set subitem1 dynamic, it also has the effect that its superset item becomes a static set and from then on its membership is frozen. The first two assignments each add one new element to subitem1. Note that both are also elements of item, as required. The third assignment is an example of the familiar indexed assignment: subitem2 is assigned all the members of item. The last assignment removes the label 'perfume' from the dynamic set subitem2. The output generated by the display statement is shown below:
Note that even though the labels 'pen' and 'pencil' were declared to be members of the set subitem1 before the assignment statements that added the labels 'ink' and 'lipstick' to the set, they appear in the listing above at the end. The reason is that elements are displayed in the internal order, which in this case is the order specified in the declaration of the set item.

Alternatively, the elements of the set subitem2 could be assigned in the following way:

```
subitem2(item) = no;
subitem2(subitem1) = yes;
subitem2('dish') = yes;
```

The first statement removes all elements from the set subitem2. The second statement adds all elements of the set subitem1. Note that this assignment is permitted since the set subitem1 is a proper subset of the set item (which is the domain of subitem2). The third statement adds one additional element.

### 4.24.2.3 Dynamic Sets with Multiple Indices

As mentioned earlier, dynamic sets may be multi-dimensional. The following lines continue the example above and illustrate assignments for multi-dimensional sets.

```
Sets sold(item) "items sold" / pencil, pen /
sup "suppliers" / bic, parker, waterman /
supply(sold, sup) ;

supply('pencil', 'bic') = yes ;
supply('pen', sup) = yes ;
```

Note that supply is a two-dimensional dynamic set. It links sold items with their respective suppliers. Other examples with multi-dimensional dynamic sets are in subsections Dynamic Sets in Conditional Assignments and Conditional Indexed Operations with Dynamic Sets below.

All the mechanisms using asterisks and parenthesized lists that were introduced in section Multi-Dimensional Sets in chapter Set Definition are available for dynamic sets as well.
4.24 Dynamic Sets

4.24.2.4 Equations Defined over the Domain of Dynamic Sets

Generally, dynamic sets are not permitted as domains in declarations of sets, variables, parameters and equations. However, they may be referenced and sometimes it is necessary to define an equation over a dynamic set.

Note

The trick is to declare the equation over the entire domain but define it over the dynamic set.

For example, defining an equation over a dynamic set can be necessary in models that will be solved for arbitrary groupings of regions simultaneously. We assume there are no explicit links between regions, but that we have a number of independent models with a common data definition and common logic. We illustrate with an artificial example, leaving out lots of details.

Set allr "all regions" / N, S, W, E, N-E, S-W /
   r(allr) "region subset for particular solution"
   type "set for various types of data" ;
Scalar price /10/ ;
Parameter revenue(allr);
Table data(allr,type) "all other data ..." ;
Variables activity1(allr) "first activity"
   activity2(allr) "second activity"
   revenue(allr) "revenue" ;
Equations resource1(allr) "first resource constraint ..."
   prodbal1(allr) "first production balance ..." ;
   resource1(r).. activity1(r) =l= data(r,'resource-1');
   prodbal1(r).. activity2(r)*price =e= revenue(r) ;

To repeat the important point: the equation is declared over the set allr, but defined over r, a subset. Note that the variables and data are declared over allr but referenced over r. Then the set r may be assigned arbitrary combinations of elements of the set allr, and the model may be solved any number of times for the chosen groupings of regions.

4.24.2.5 Assigning Membership to Singleton Sets

Singleton sets have only one element. Hence any assignment to a singleton set first clears or empties the set, no explicit action to clear the set is necessary. This is illustrated with the following example:

Set    i   "Static Set"   / a, b, c /
   ii(i) "Dynamic Set"   / b /;
Singleton Set si(i) "Dynamic Singleton Set" / b /;
ii('c') = yes;
si('c') = yes;
display ii, si;

Note that both ii and si are subsets of the set i, but only si is declared as a singleton set. The assignment statements assign to both sets the element 'c'. While 'c' is added to the set ii, it replaces the original element in the singleton set si. The output from the display statement confirms this:
---- 8 SET ii Dynamic Set
b, c

---- 8 SET si Dynamic Singleton Set
c

For more information on singleton sets in GAMS, see section Singleton Sets.

Attention

That an assignment to a singleton set first clears the set always, means that it is even cleared if there would be no change at all for a regular set:

Singleton Set s / 1 /;
s(s)$0 = yes;
display s;

Here is the output from the display statement in the listing file:

---- 3 SET s

( EMPTY )

The assignment behavior can be changed with the option and command line parameter strictSingleton which affects the behavior of a membership assignment to a Singleton Set. With strictSingleton=0 GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With strictSingleton=1 (default), such an assignment raises an error. Consider the following example:

Set i "Static Set" / a, b, c /
Singleton Set si(i) "Dynamic Singleton Set";
si(i) = ord(i) > 1;
display si;

By default, the above code will trigger an error as an assignment to a singleton set with more than one element on the right hand side is forbidden:

*** Error at line 3: Multiple assignment to Singleton Set not allowed (see option strictSingleton)

However, with option (or command line parameter) strictSingleton=0 GAMS does not complain about such an assignment with more than one element on the right hand side but takes the first one:

Set i "Static Set" / a, b, c /
Singleton Set si(i) "Dynamic Singleton Set";
option strictSingleton = 0;
si(i) = ord(i) > 1;
display si;

The output from the display statement confirms this:

---- 5 SET si Dynamic Singleton Set
b

4.24.3 Set Operations

GAMS provides symbols for arithmetic set operations that may be used with dynamic sets. An overview of the set operations in GAMS is given in Table 1. Examples and alternative formulations for each operation follow. Note that in the table below the set i is the static superset and the sets j and k are dynamic sets.
4.24 Dynamic Sets

<table>
<thead>
<tr>
<th>Set Operation</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Union</td>
<td>j(i) + k(i)</td>
<td>Returns a subset of ( i ) that contains all the elements of the sets ( j ) and ( k ).</td>
</tr>
<tr>
<td>Set Intersection</td>
<td>j(i) * k(i)</td>
<td>Returns a subset of ( i ) that contains the elements of the set ( j ) that are also elements of the set ( k ).</td>
</tr>
<tr>
<td>Set Complement</td>
<td>not j(i)</td>
<td>Returns a subset of ( i ) that contains all the elements of the set ( i ) that are not elements of the set ( j ).</td>
</tr>
<tr>
<td>Set Difference</td>
<td>j(i) - k(i)</td>
<td>Returns a subset of ( i ) that contains all the elements of the set ( j ) that are not elements of the set ( k ).</td>
</tr>
</tbody>
</table>

**Table 1:** Set Operations with Dynamic Sets

The following examples draw on the database model [ZLOOF] that we introduced above. Recall that the set \( \text{item} \) is the superset of the dynamic sets \( \text{subitem1} \) and \( \text{subitem2} \). We add new dynamic sets for the results of the respective set operations. The following example illustrates that the dynamic set operations are equivalent to the following alternative ways of representation.

Sets \( \text{union1(item)}, \text{intersection1(item)}, \text{complement1(item)}, \text{difference1(item)} \)

\[
\text{union1(item)} = \text{subitem1(item)} + \text{subitem2(item)}; \quad \text{display union1;}
\]

\[
\text{union2(item)} = \text{subitem1(item)}; \quad \text{display union2;}
\]

\[
\text{intersection1(item)} = \text{subitem1(item)} * \text{subitem2(item)}; \quad \text{display intersection1;}
\]

\[
\text{intersection2(item)} = \text{yes$(subitem1(item) and subitem2(item))}$; \quad \text{display intersection2;}
\]

\[
\text{complement1(item)} = \text{not subitem1(item)}; \quad \text{display complement1;}
\]

\[
\text{complement2(item)} = \text{yes; complement2(subitem1) = no}; \quad \text{display complement2;}
\]

\[
\text{difference1(item)} = \text{subitem2(item)} - \text{subitem1(item)}; \quad \text{display difference1;}
\]

\[
\text{difference2(item)} = \text{yes$(subitem2(item))}$; \quad \text{display difference2;}
\]

The display statements will show that the above assignment statements for each operation result in the same dynamic set like using the set operator. Observe that the alternative formulations for the set intersection and set difference involve conditional assignments. Conditional assignments in the context of dynamic sets are discussed in depth in the next section.

**Note**

The indexed operation `sum` may be used for set unions. Similarly, the indexed operation `prod` may be used for set intersections. For examples see section **Conditional Indexed Operations with Dynamic Sets** below.

### 4.24.4 Using Dollar Controls with Dynamic Sets

The remainder of this chapter assumes familiarity with the **dollar condition** that is introduced in chapter **Conditional Expressions, Assignments and Equations**. All the dollar control machinery is available for use with dynamic sets. In fact, the full power of dynamic sets can be exploited using these dollar controls.

Recall that set membership of subsets and dynamic sets may be used as a logical condition; see subsection **Logical Conditions: Set Membership and Set Functions** for details. Set membership may also be a building block in complex logical conditions that are constructed using the logical operators `not`, `and`, `or`, `xor`, `imp` and `eqv`. Moreover, the set operations introduced in the previous section may also be used in logical conditions. Like other dollar conditions, dollar conditions with dynamic sets are used in the context of assignments, indexed operations and equations. We will discuss in detail each of these in the following subsections.

Apart from being part of logical conditions, dynamic sets may be assigned members with conditional assignments. Examples are given in the next subsection.
4.24.4.1 Dynamic Sets in Conditional Assignments

Dynamic sets may be used in two ways in conditional assignments: they may be the item on the left-hand side that is assigned to and they may be part of the logical condition. Below we present examples for both. The examples are again based on the database model [ZLOOF] that we introduced above.

```
Set item "all items" / dish, ink, lipstick, pen, pencil, perfume /
   subitem1(item) "first subset of item" / ink, lipstick, pen, pencil /
   subitem2(item) "second subset of item";
subitem2(item)$subitem1(item) = yes;
display subitem2;
```

The conditional assignment adds the members of dynamic set subitem1 to the dynamic set subitem2. Thus subitem2 will have the following elements:

```
---- 6 SET subitem2  second subset of item
ink , lipstick, pen , pencil
```

Note that instead of using subitem1 in a dollar condition we could also write:

```
subitem2(subitem1) = yes;
```

In the next example of a conditional assignment, a dynamic set features in the logical condition on the right-hand side. The first statement clears the set subitem2 of any previously assigned members and the second statement assigns all members of subitem1 to subitem2. The following conditional assignment will have the same result:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item);
```

The logical condition in this assignment is subitem1(item). It is satisfied for all members of the set subitem1. Hence the statement assigns all elements of the domain item that are members of the set subitem1 to the dynamic set subitem2. Note that in this assignment the dollar operator is on the right. In the section Dollar on the Right we show that conditional assignments with the dollar operator on the right-hand side imply an if-then-else structure where the else case is automatically zero. Unlike parameters, dynamic sets cannot be assigned the value of zero, they are assigned the value no instead. Therefore a more explicit formulation of the conditional assignment above would be:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item) + no$(not subitem1(item));
```

For more on sets in logical conditions, see section Logical Conditions: Set Membership and Set Functions. For more on conditional assignments, see section Conditional Assignments.
4.24 Dynamic Sets

4.24.4.2 Conditional Indexed Operations with Dynamic Sets

Indexed operations in GAMS are introduced in section Indexed Operations. They may be controlled by dollar conditions as discussed in section Conditional Indexed Operations. The domain of conditional indexed operations is often restricted by a set, called the conditional set. Dynamic sets may be used as conditional sets or they may be assigned to with a statement that features a conditional indexed operation on the right-hand side. We will illustrate both cases with examples.

Suppose we have a set of origins, a set of destinations and a table specifying the flight distance between them:

```gams
Set i "origins" / Chicago, Philadelphia /
    j "destinations" / Vancouver, Bogota, Dublin, Rio, Marrakech / ;
Table d(i,j) "distance (miles)"
    Vancouver  Bogota  Dublin  Rio    Marrakech
Chicago      1777    2691    3709   5202   4352
Philadelphia 2438    2419    3306   4695   3757 ;
```

We wish to find the longest distance that we can travel given that we have a limit of 3500 miles.

```gams
Set can_do(i,j) "connections with less than 3500 miles";
can_do(i,j)$(d(i,j) < 3500) = yes;
display can_do;
Scalar maxd "longest distance possible"
maxd = smax((i,j)$can_do(i,j), d(i,j));
display maxd;
```

The dynamic set `can_do` contains all connections that are less than 3500 miles. The scalar `maxd` is defined by a conditional assignment where the indexed operation `smax` scans all entries of the parameter `d` whose label combinations are members of the set `can_do` and chooses the largest value. The output generated by the `display` statements is shown below:

```
---- 11 SET can_do connections with less than 3500 miles

<table>
<thead>
<tr>
<th></th>
<th>Vancouver</th>
<th>Bogota</th>
<th>Dublin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Philly</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

---- 15 PARAMETER maxd = 3306.000 longest distance possible
```

There is a shorter alternative formulation for this assignment; see subsection Filtering through Dynamic Sets below for details.

Finally, we also wish to know which flight connection is linked to the longest possible distance. Consider the following two lines:

```gams
Singleton set maxc(i,j) "maximum distance connection";
maxc(i,j) = yes$can_do(i,j) and (d(i,j) = maxd);
```

---
The dynamic singleton set is assigned the member of the dynamic set can_do whose distance equals the maximum distance.

The full power of indexed operators becomes apparent with multi-dimensional dynamic sets. As earlier in this chapter, we illustrate with fragments of code adapted from the relational database model [ZLOOF].

Set dep "departments" / cosmetics, hardware, household, stationary, toy /
sup "suppliers" / bic, dupont, parker, revlon /
item "items sold" / dish, ink, lipstick, pen, pencil, perfume /
sales(dep,item) "departments and items sold" /
    cosmetics. (lipstick,perfume)
    hardware. ink
    household. (dish,pen)
    stationary. (dish,ink,pen,pencil)
    toy. (ink,pen,pencil) /
supply(item,sup) "items and suppliers" /
    dish.(bic,dupont), ink.(bic,parker), lipstick.revlon
    pen.(parker,revlon), pencil.(bic,parker), perfume.revlon /
Set g03(dep) "departments selling items supplied by Parker";
g03(dep) = sum(item$supply(item,'parker'), sales(dep,item));
display g03;

The assignment above is used to create the set of departments that sell items supplied by 'parker'. Note that the set g03 is a subset of the set dep. Its members are specified by assignment, hence it is a dynamic set. Note that the assignment is made to a set, therefore the indexed operator sum refers to a set union (and not to an addition as would be the case if the assignment were made to a parameter). The indexed operation is controlled by the two-dimensional set supply with the label 'parker' in the second index position. This logical condition is TRUE for all members of the set supply where the second index is 'parker'. Hence the summation is over all items sold, provided that the supplier is 'parker'. Given the declaration of the set supply, this means 'ink', 'pen' and 'pencil'. The associated departments are thus all departments except for 'cosmetics':

---- 19 SET g03 departments selling items supplied by Parker
    hardware , household , stationary , toy

Now suppose we are interested in the departments that are selling only items supplied by 'parker'. We introduce a new dynamic set g11 and the following assignment adds the desired departments:

Set g11(dep) "departments only selling items supplied by parker";
g11(dep) = prod(sales(dep,item), supply(item,"parker"));
display g11;

Note that the indexed operation prod refers to set intersections in the context of assignments to dynamic sets. From all departments linked with items only those are included where all items sold are supplied by 'parker'. This means that departments that additionally sell items that are not supplied by 'parker' are excluded. Hence, only 'hardware' and 'toy' are added to g11.
4.24.4.3 Conditional Equations with Dynamic Sets

Recall that dollar conditions in the context of equations may restrict the domain of the equation and they may also feature in the algebraic formulation of the equation; see section Conditional Equations for more information. In both instances dynamic sets may be used as part of the logical condition. Dollar conditions with dynamic sets in the algebra of equations are similar to conditional assignments with dynamic sets; see section Dynamic Sets in Conditional Assignments above. The example that follows illustrates the use of a dynamic set to restrict the domain of definition of an equation. In section Equations Defined over the Domain of Dynamic Sets above we had the following equation definition:

\[
\text{prodbal1}(r) \ldots \quad \text{activity2}(r) \cdot \text{price} = \text{e} = \text{revenue}(r) ;
\]

Recall that \( r \) is a dynamic set and a subset of the set \( \text{allr} \). Hence this equation may be rewritten in the following way:

\[
\text{prodbal1}(\text{allr}) \cdot \$ \cdot \text{r(\text{allr})} \ldots \text{activity2(\text{allr})} \cdot \text{price} = \text{e} = \text{revenue(\text{allr})} ;
\]

Note that both formulations achieve the same result: restricting the domain of definition to those elements that belong to the dynamic set \( r \). While in the second formulation the condition is specified explicitly, in the first formulation the domain is *filtered* through the dynamic set \( r \). This is the topic of the next subsection.

4.24.4.4 Filtering through Dynamic Sets

The filtering process is introduced and explained in section Filtering Sets in chapter Conditional Expressions, Assignments and Equations. In certain circumstances it is an alternative to the dollar condition to restrict the domain of equations, sets, variables, parameters and indexed operations. We already saw an example for restricting the domain of definition of an equation in the previous subsection. The next example refers to restricting the domain in an indexed operation. In section Conditional Indexed Operations with Dynamic Sets we had the following assignment:

\[
\text{maxd} = \text{smax}((\text{i}, \text{j}) \cdot \$ \cdot \text{can_do(i), j}), \text{d(i, j)});
\]

Recall that \( \text{maxd} \) is a scalar, \( \text{i} \) and \( \text{j} \) are sets, \( \text{can_do} \) is a dynamic set and \( \text{d} \) is a two-dimensional parameter. Note that the conditional set is the dynamic set \( \text{can_do} \). The assignment may be rewritten in the following way:

\[
\text{maxd} = \text{smax}((\text{can_do(i, j)}), \text{d(i, j)}) ;
\]

Here the indexed operation is filtered through the dynamic set \( \text{can_do} \), a dollar condition is not necessary.
4.25 Sets as Sequences: Ordered Sets

4.25.1 Introduction

Sets are introduced in chapter Set Definition. There we state that in general, sets in GAMS are regarded as an unordered collection of labels. However, in some contexts, say, multi-period planning models, some sets need to be treated as if they were sequences. In this chapter we will establish the notion of ordered sets and we will cover their special features and the associated operations.

Examples where ordered sets are needed include economic models that explicitly represent conditions in different time periods that are linked, location problems where the formulation may require a representation of contiguous areas, as in a grid representation of a city, scheduling problems and programs that model stocks of capital with equations of the form \( \text{stocks at the end of period } n \text{ are equal to stocks at the end of period } n - 1 \text{ plus net gains during period } n \).

Note
Models involving sequences of time periods are often called dynamic models, because they describe how conditions change over time. This use of the word dynamic unfortunately has a different meaning from that used in connection with dynamic sets, but this is unavoidable.

4.25.2 Ordered and Unordered Sets

Certain one-dimensional sets may be treated as if they were a sequence. Those sets need to be ordered and static. A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the labels in the GAMS Entry order.

Note
- The GAMS entry order is the order in which the individual labels first appear in the GAMS program, either explicitly or as a result of using the shorthand asterisk notation.
- For the sake of simplicity, sets that are static and ordered are often just referred to as ordered sets.

GAMS maintains a unique element list where all labels that are used as elements in one or more sets are listed. The order of the elements in any one set is the same as the order of those elements in the unique element list. This means that the order of a set may not be what it appears to be if some of the labels were used in an earlier definition. The internal GAMS order of the labels can be made visible with the dollar control option \$onUELlist. This directive generates a map that is shown in the compilation output of the listing file. For details on the listing file and GAMS output in general, see chapter GAMS Output. A good rule of thumb is that if the user wants a set to be ordered and the labels in the set have not been used already, then they will be ordered.

In the example below we show ordered and unordered sets and the map showing the order. The input is:

\[
\text{$onUELlist}
\]
Note that the label "1987" is the first label seen by GAMS. It appears again as the last label in the initialization list for the set t2. This means that the set t2 is not ordered and any attempt to use t2 in a context implying order, such as the operations introduced later in this chapter, will cause error messages. Observe that the set t3 is ordered, as all the members of t3 have appeared in the GAMS program before, and in the same order that they are listed in the definition of t3.

The unique element listing map below shows the entry order (the important one) and the sorted order, obtained by sorting the labels into dictionary order. The single digits on the left are the sequence numbers of the first label on that line.

<table>
<thead>
<tr>
<th>Entry Order</th>
<th>Sorted Order</th>
</tr>
</thead>
</table>

A set can always be made ordered by moving its declaration closer to the beginning of the program. With these restrictions in mind, we will continue in the next section with the operations that may be used with ordered sets.

**4.25.3 Sorting a Set**

Note that besides the entry order of unique elements there is also a sorted (alphabetical) order. To obtain the sorted order (of an ordered or unordered set) the special predefined two-dimensional set SortedUELs(∗,∗) can be used. Consider the following example where set j is reported in an alphabetically sorted format:

```gams
define set j / c, a, b, 1, 2, 11 /;
define display j;
alias(∗, u);
define file ordered /ordered.txt/;
define loop(SortedUEL(u, j), put ordered j.tl:0 ' ');
putclose ordered;
```

The output generated by the display statement follows:

```
-----          2 SET j
  c,  a,  b,  1,  2,  11
```

Note that, as expected, the display statement shows the elements of set j in the entry order not in alphabetical order. However, the elements are listed in alphabetical order in the file ordered.txt. Note furthermore, that alphabetical sorting leads to an order where e.g. 11 precedes 2.

```
1 11 2 a b c
```

In the example above u is aliased with the Universal Set. For an introduction to writing external files with put, see chapter The Put Writing Facility.
4.25.4 Ord and Card Operators

As stated in section Labels in chapter GAMS Programs, labels in GAMS do not have a numerical value. The examples used were that the label '1986' does not have a numerical value of 1986 and the label '01' is different from the label '1'. This section introduces two operators - ord and card - that return integer values when applied to sets. While the integer values returned do not represent the numerical value of the label, they can be used for the same purpose.

Note

GAMS provides some string manipulation capability by extending the card and ord operators to also work on strings.

4.25.4.1 The Ord Operator

The operator ord returns the relative position of a member in a set.

Attention

By default the operator ord may be used only with one-dimensional sets that are static and ordered.

Some examples show the usage.

Set t "time periods" / 1985*1995 / ;
Parameter val(t) ;
val(t) = ord(t);

Note that as a result of the statements above, the value of val('1985') will be 1, val('1986') will be 2 and so on.

A common use of ord is in setting up vectors that represent quantities growing in some analytically specified way. For example, suppose a country has 56 million people in the base period and the population is growing at the rate of 1.5 percent per year. Then the population in succeeding years can be calculated as follows:

population(t) = 56*(1.015**(ord(t) - 1)) ;

It is often useful to simulate general matrix operations in GAMS. The first index of a two dimensional parameter can conveniently represent the rows, the second the columns and order is necessary. The example below shows how to set the upper triangle of a matrix equal to the row index plus the column index, and the diagonal and lower triangle to zero.

Set i "row and column labels" / x1*x10 /;
alias (i,j);
Parameter a(i,j) "a general square matrix";
a(i,j) $(ord(i) < ord(j)) = ord(i) + ord(j);
Note that in the assignment statement the logical condition \((\text{ord}(i) < \text{ord}(j))\) restricts the assignment to the entries of the upper triangle. For more information on logical conditions and conditional assignments in GAMS, see sections Logical Conditions and Conditional Assignments respectively.

Note that the strict requirement that the set needs to be ordered for the operator \(\text{ord}\) to be used may be relaxed with a dollar control option called \text{offOrder}. Consider the following lines where we revisit the example from the previous section.

\[
\text{\$offOrder}
\]

\[
\]

\[
\]

\[
\text{Parameter p(t2);}\]

\[
\text{p(t2) = \text{ord}(t2);}\]

\[
\text{display p;}
\]

Note that, as we explained above, the set \(t2\) is not ordered. Therefore using \(\text{ord}(t2)\) somewhere in the model will usually cause the program to be terminated with an error message. However, with the dollar control option \text{offOrder} active, the set \(t2\) is treated as if it were ordered and the display statement will generate the following output:

\[
\begin{align*}
\text{---- 6 PARAMETER p} \\
\text{1987 1.000, 1983 2.000, 1984 3.000, 1985 4.000, 1986 5.000}
\end{align*}
\]

While this may be useful in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the \(\text{ord}\) operator can be turned on again via \text{onOrder}.

### 4.25.4.2 The Card Operator

The operator \text{card} returns the number of elements in a set.

Note

The operator \text{card} may be used with any set: static and dynamic sets, ordered and unordered sets.

The following example illustrates its use:

\[
\text{Set \ t "time periods" / 1985*1995 /;}
\]

\[
\text{Scalar s;}
\]

\[
\text{s = \text{card}(t);}\]

\[
\text{Display s;}
\]

Note that \(s\) will be assigned the value 11 since the set \(t\) has 11 elements.

A common use of \text{card} is to specify some condition only for the final period, for example to fix a variable. Consider the following artificial example:

\[
\text{c.fx(t)$(\text{ord}(t) = \text{card}(t)) = \text{demand}(t);}\]

Note that the logical condition \((\text{ord}(t) = \text{card}(t))\) restricts the assignment to the last element of the set \(t\): no assignment is made for other members of \(t\). The advantage of this way of fixing the variable \(c\) is that the membership of \(t\) can be changed safely and this statement will always fix \(c\) for the last element. For more information on logical conditions and conditional assignments in GAMS, see sections Logical Conditions and Conditional Assignments respectively.

### 4.25.5 Lag and Lead Operators

Lag and lead operators are used to relate the \textit{current} member of an ordered set to the \textit{previous} or \textit{next} member of the set. GAMS provides two forms of lag and lead operators (linear and circular), they are summarized in Table 1. Note that in the table below \(t\) is a member of an ordered set and \(n\) is a positive integer.
Table 1: Linear and Circular Lag and Lead Operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Lag</td>
<td>t - n</td>
<td>Refers to the element of the ordered set whose relative position in the set is ( \text{ord}(t) - n ).</td>
</tr>
<tr>
<td>Linear Lead</td>
<td>t + n</td>
<td>Refers to the element of the ordered set whose relative position in the set is ( \text{ord}(t) + n ).</td>
</tr>
<tr>
<td>Circular Lag</td>
<td>t -- n</td>
<td>Same as ( t - n ), only here the first element of the set is assumed to be preceded by the last element of the set, thus forming a circle of elements.</td>
</tr>
<tr>
<td>Circular Lead</td>
<td>t ++ n</td>
<td>Same as ( t + n ), only here the last element of the set is assumed to be followed by the first element of the set, thus forming a circle of elements.</td>
</tr>
</tbody>
</table>

Note that the only difference between linear and circular lag and lead operators is how endpoints are treated. Linear operators assume that there are no members preceding the first and following the last member of the ordered set. This assumption may result in elements of the set being referenced that actually do not exist. Therefore the user must think carefully about the treatment of endpoints: models with linear lag and lead operators will need special exception handling logic to deal with them. The next two sections will describe how this issue is handled in GAMS in the context in which these operators are typically used: assignments and equations. Linear lag and lead operators are useful for modeling time periods that do not repeat, like a set of years (say 1990 to 1997).

Circular lag and lead operators assume that the first and last members of the set are adjacent, so as to form a circular sequence of members. This means that \( \text{first}--1 \) is a reference to \( \text{last} \) and \( \text{last}++2 \) is the same as \( \text{first}++1 \). All references and assignments are defined. The assumption of circularity is useful for modeling time periods that repeat, such as months of the year or hours in the day. It is quite natural to think of January as the month following December. Agricultural farm budget models and workforce scheduling models are examples of applications where circular leads occur naturally.

Note

- GAMS is able to distinguish the linear lag and lead operators \('-\) and \('+\)' from arithmetic operators by context. To avoid ambiguity, GAMS does not allow to mix lag and lead operators with arithmetic operators. For example, \( i+1+1 \) is not allowed, but writing \( i+(1+1) \) would work.
- Observe that the lag (or lead) value \( n \) does not have to be an explicit constant: it can be an arbitrary expression, provided that it evaluates to an integer. If it does not, error messages will be produced. A negative result causes a switch in sense (from lag to lead, for example).

Note that if lag and lead operators are used with an unordered set, the program will terminate with an error message. The strict requirement that the set be ordered may be relaxed with the dollar control option \$offOrder. If the directive \$offOrder is added, in the lines that follow unordered sets are treated as if they were ordered and therefore lag and lead operators may be used with them. While this may be advantageous in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the use of lag and lead operators can be turned on again via \$onOrder.

The next two subsections will give some illustrative examples on the use of lag and lead operators in assignment statements and in equations respectively.
4.25.5.1 Lags and Leads in Assignments

Lag and lead operators may be used in assignment statements. The use of a lag or lead operator on the right-hand side of an assignment is called a reference, while their use on the left-hand side is called an assignment and involves the definition of a domain of the assignment. The concepts behind reference and assignment are equally valid for the linear and circular forms of the lag and lead operators. However, the importance of the distinction between reference and assignment is not pronounced for circular lag and lead operators, because non-existent elements are not referred to in this case.

Note

A reference to a non-existent element causes the default value zero to be used, whereas an attempt to assign to a non-existent element results in no assignment being made.

We will first illustrate linear lag and lead operators for reference and assignment. Then we will turn to the circular form of the operators.

4.25.5.1.1 Linear Lag and Lead Operators in Assignments - Reference

Consider the following example, where the lag operator is used on the right-hand side of an assignment statement:

Set t "time sequence" / y-1987*y-1991 /;
Parameter a(t), b(t);
a(t) = 1986 + ord(t);
b(t) = -1;
b(t) = a(t-1);
option decimals = 0;
display a, b;

Note that the option statement suppresses the decimal places from the display statement.

---- 7 PARAMETER a
---- 7 PARAMETER b

Note that, as expected, the values for the parameter a are 1987, 1988 up to 1991 corresponding to the labels "y-1987", "y-1988" and so on. Observe that the parameter b is initialized to −1 so that the result of the next assignment can be seen clearly. Note that in the last assignment the lag operator '−' is used on the right-hand side, resulting in the values for b to equal the values for a from the previous period. If there is no previous period, as with the first element, "y-1987", the value zero is assigned, replacing the previous value of −1 (values of zero for parameters are not displayed).
4.25.5.1.2 Linear Lag and Lead Operators in Assignments - Assignment  Consider the following variation of the previous example. Here the lead operator is used on the left-hand side of an assignment statement:

Set t "time sequence" / y-1987*y-1991 / ;
Parameter a(t), c(t) ;
a(t) = 1986 + ord(t) ;
c(t) = -1;
c(t+2) = a(t) ;
option decimals = 0;
display a, c ;

Note that, as before, the option statement suppresses the decimal places from the display statement.

---- 7 PARAMETER a
---- 7 PARAMETER c

The assignment to a is explained in the previous subsection. Note that the parameter c is initialized to -1. The assignment to c involving the lead operator on the left-hand side needs special attention. It is best to spell out step by step how this assignment is made. For each member of t in sequence, find the member of c associated with t+2. If it exists, replace its value with the value of a(t). If not (as with labels "y-1990" and "y-1991") make no assignment. The first member of the set t is "y-1987", therefore the first assignment is made to c("y-1989") which takes the value of a("y-1987"), that is 1987. No assignments at all are made to c("y-1987") and c("y-1988"); these two retain their previous values of -1.

4.25.5.1.3 Circular Lag and Lead Operators in Assignments  The following example illustrates the use of circular lag and lead operators in assignment statements.

Set s "seasons" / spring, summer, autumn, winter /;
Parameter val(s) / spring 10, summer 15, autumn 12, winter 8 /
    lagval(s)
    leadval(s);
lagval(s) = -1 ;
lagval(s) = val(s--2) ;
leadval(s) = -1 ;
leadval(s++1) = val(s) ;
option decimals = 0;
display val, lagval, leadval;

Note that, as before, the option statement suppresses the decimal places from the display statement. The results are shown below.

----- 10 PARAMETER val
spring 10, summer 15, autumn 12, winter 8
----- 10 PARAMETER lagval

---
In the example parameter lagval is used for reference while leadval is used for assignment. Notice that the case of circular lag and lead operators does not refer to any non-existent elements. The difference between reference and assignment is therefore not important. Note that the following two statements from the example above,

\[
\begin{align*}
\text{lagval}(s) & = \text{val}(s--2) ; \\
\text{leadval}(s++1) & = \text{val}(s) ;
\end{align*}
\]

are equivalent to

\[
\begin{align*}
\text{lagval}(s++2) & = \text{val}(s) ; \\
\text{leadval}(s) & = \text{val}(s--1) ;
\end{align*}
\]

The use of reference and assignment have been reversed with no difference in effect.

### 4.25.5.2 Lags and Leads in Equations

The principles established in the previous section follow quite naturally into equation definitions. A lag or lead to the left of the ‘..’ symbol is a modification of the domain of definition of the equation. The linear form may cause one or more individual equations to be suppressed. A lag or lead operation in the body of an equation (to the right of the ‘..’ symbol) is a reference. If the associated label is not defined, the term vanishes.

Note

All lag and lead operands must be **exogenous**. For more information, see section **Functions in Equation Definitions**.

In the next two subsections we will provide examples illustrating the use of the linear form of the lag and lead operators in equations to modify the domain of definition and for reference respectively. In the last subsection we will turn to circular lag and lead operators in equations.

### 4.25.5.2.1 Linear Lag and Lead Operators in Equations - Domain Control

Consider the following simple artificial multi-period example. We specify a complete model and encourage users to solve it and further explore it.
Sets t / t1*t5 / tfirst(t);
Parameter i(t) / #t 1 /;
Scalar k0 / 3.00 /;

tfirst(t) = yes$(ord(t) = 1);

Variables k(t), z;
k.fx(tfirst) = k0;

Equations kk(t), dummy;
dummy.. z =e= 0;
kk(t+1).. k(t+1) =e= k(t) + i(t);

model m /all/;
option limrow = 10;
solve m using lp min z;

Note that the equation \(kk\) is declared over the set \(t\), but it is defined over the domain \((t+1)\). Therefore the first equation that will be generated is the following:

\[
k('t2') =e= k('t1') + i('t1');
\]

Note that the value of the variable \(k('t1')\) is fixed at the value of scalar \(k0\). Observe that for the last element of \(t\), the term \(k(t+1)\) is not defined and therefore the equation will not be generated. If lag or lead operators are used in the domain of definition of an equation, the equation listing can be a useful tool to verify whether the equations that have been generated are those that were intended:

\[
\begin{align*}
kk(t2) &.. - k(t1) + k(t2) =E= 1 ; (LHS = 0, INFES = 1 ****) \\
nk(t3) &.. - k(t2) + k(t3) =E= 1 ; (LHS = 0, INFES = 1 ****) \\
nk(t4) &.. - k(t3) + k(t4) =E= 1 ; (LHS = 0, INFES = 1 ****) \\
nk(t5) &.. - k(t4) + k(t5) =E= 1 ; (LHS = 0, INFES = 1 ****)
\end{align*}
\]

To summarize, the lead operator in the domain of definition has restricted the number of constraints generated so that there are no references to non-existent variables.

For a more realistic model that illustrates the usage of linear lag operators in equations, see for example the optimal economic growth model [RAMSEY].

### 4.25.5.2.2 Linear Lag and Lead Operators in Equations - Reference

In the previous subsection we showed how to write the equation \(kk\) using the lead operator for domain control in combination with fixing the variable \(k(tfirst)\) to \(k0\). An alternative formulation could neglect the fixing of \(k(tfirst)\) and use a lag operator and a dollar condition in the body of the equation while the domain of definition is unrestricted:

\[
kk(t). \ k(t) =e= k(t-1) + i(t-1) + k0$tfirst(t);
\]

Note that for the first element of the set \(t\) the terms \(k(t-1)\) and \(i(t-1)\) are not defined and therefore vanish. Without the conditional term, the resulting equation would be:

\[
k('t1') =e= 0;
\]

However, this would lead to different results as \(k('t1')\) would not be set to the value of \(k0\) anymore. Therefore the conditional expression \(k0$tfirst(t)\) is added. Observe that in this formulation equations are generated for all time periods, no equation is suppressed.

In general, the choice between using lag and lead operators as reference like in the last example or in domain control is often a matter of taste.
In the case of circular lag and lead operators, the difference between their use in domain control and as reference is not important because it does not lead to any equations or terms being suppressed. Consider the following artificial example.

```
Set s "seasons" / spring, summer, autumn, winter /;

Variable produ(s) "amount of goods produced in each season"
      avail(s) "amount of goods available in each season"
      sold(s) "amount of goods sold in each season";

Equation matbal(s);

matbal(s).. avail(s++) =e= avail(s) + produ(s) - sold(s);
```

In this example four individual equations are generated. They are listed below.

```
avail('summer') =e= avail('spring') + produ('spring') - sold('spring');
avail('autumn') =e= avail('summer') + produ('summer') - sold('summer');
avail('winter') =e= avail('autumn') + produ('autumn') - sold('autumn');
avail('spring') =e= avail('winter') + produ('winter') - sold('winter');
```

Note that for the last element of the set s the term avail(s++) is evaluated to avail('spring'). This term is well defined and therefore it does not vanish. Similarly, using the circular lead operator in the domain of definition like in the following line will result in the same four equations being generated as above and no equation being suppressed.

```
matbal(s++).. avail(s++) =e= avail(s) + produ(s) - sold(s);
```

4.25.6 Summary

This chapter introduced the concept of ordered sets. All the features in GAMS that dealt with this issue including the ord and card functions, as well as the linear and circular forms of the lag and lead operators were described in detail.

4.26 Data Manipulations with Parameters

4.26.1 Introduction

In this chapter we explicitly cover only parameter manipulation, including all aspects of data handling. Much of this material is relevant elsewhere (e.g., to sets), but for specifics related to assignment to sets, to conditions, and to assignments within flow control constructs such as the loop statement, see chapters Dynamic Sets, Conditional Expressions, Assignments and Equations and Programming Flow Control Features respectively.

Once initialized, data often requires manipulation in order to bring it into a form most suitable for use in a model or an application. The assignment statement is the way to do this. All the possible components of the assignment statement except conditions are introduced and discussed in this chapter.
4.26.2 The Assignment Statement

The assignment statement is the fundamental data manipulation statement in GAMS. It may be used to define or alter values associated with sets, parameters, variables or equations.

A simple assignment is written in the style associated with many other computer languages. GAMS uses the traditional symbols for addition (+), subtraction (-), multiplication (*) and division (/). We will use them in the examples that follow, and give more details in section Expressions.

4.26.2.1 Scalar Assignments

Consider the following artificial sequence:

```gams
scalar x / 1.5 /;
  x = 1.2;
  x = x + 2;
```

The scalar `x` is initialized to be 1.5. The second statement changes the value to 1.2, and the third changes it to 3.2. The second and third statements are assignments: each replaces the current value of `x` with a new one.

Note that the same symbol can be used on the left and right of the = sign. The new value is not available until the calculation is complete, and the operation gives the expected result.

Note

An assignment cannot start with a reserved word. A semicolon is therefore required as a delimiter before all assignments.

Note that extended range identifiers (e.g. `INF`) and acronyms may also be used in assignment statements. For specific details, see sections Extended Range Arithmetic and Acronym Usage respectively.

4.26.2.2 Indexed Assignments

The GAMS syntax for performing indexed assignments is extremely powerful. This operation offers what may be thought of as simultaneous or parallel assignments and provides a concise way of specifying large amounts of data. Consider the mathematical statement \( DJ_d = 2.75 \cdot DA_d \) for all elements of \( d \). This means that for every member of the set \( d \), a value is assigned to \( DJ \). This can be written in GAMS as follows:

```gams
dj(d) = 2.75*da(d) ;
```

This assignment is known technically as an indexed assignment and set \( d \) as the controlling index or controlling set.
Attention

The index set(s) on the left hand side of an indexed assignment are referred to synonymously as the controlling indices, controlling sets, or controlling domain of the assignment.

The extension to two or more controlling indices should be obvious. There will be an assignment made for each label combination that can be constructed using the indices inside the parentheses. Consider the following example of an assignment to all 100 data elements of the parameter a.

```
sets row / r-1*r-10 /
col / c-1*c-10 /
sro(row) / r-7*r-10 /;
parameters r(row) / r-1*r-7 4, r-8*r-10 5/
c(col) / c-1*c-5 3, c-6*c-10 2/;
parameters a(row,col);
a(row,col) = 13.2 + r(row)*c(col);
The calculation in the last statement is carried out for each of the 100 unique two-label combinations that can be formed from the elements of row and col. An explicit formulation of the first of these assignments follows:

a('r-1','c-1') = 13.2 + r('r-1')*c('c-1');
```

Note that for indexed assignments a copy of the symbols on the right hand side is installed before the assignment is carried out. That means it does not work "in-place" or recursively. Consider the following example where we compute the first ten Fibonacci numbers and store them in parameter f using a loop. The example also illustrates how such a recursive calculation does not work with a parallel assignment statement for parameter g.

```
set i / i1*i10 /
parameter
  f(i) / i1 1 /
g(i) / i1 1 /;
loop(i$(ord(i)>=2),
  f(i) = f(i-2) + f(i-1);
);
g(i)$$(ord(i)>=2) = g(i-2) + g(i-1)
display f,g;
The display statement results in the following output.

---- 9 PARAMETER f
  i1 1.000,   i2 1.000,   i3 2.000,   i4 3.000,   i5 5.000
  i6 8.000,   i7 13.000,  i8 21.000,  i9 34.000, i10 55.000

---- 9 PARAMETER g
  i1 1.000,   i2 1.000,   i3 1.000
```
4.26.2.3 Restricting the Domain in Assignments

Sometimes it is necessary to make assignments over selected elements of a set instead of over the entire domain. There are several ways to accomplish this: using explicit labels, subsets, conditionals and tuples.

4.26.2.3.1 Restricting the Domain: Explicit Labels  The strongest restriction of the domain is assigning a value to just one element. Labels may be used explicitly in the context of assignments to accomplish this. The following example illustrates:

\[ a(’r-7’,’c-4’) = -2.36; \]

This statement assigns a constant value to just one element of the parameter \( a \). All other elements of \( a \) remain unchanged. Labels must be quoted when used in this way.

4.26.2.3.2 Restricting the Domain: Subsets  In general, wherever a set name may occur in an indexed assignment, a subset may be used instead.

Consider the following example:

\[ a(sro,’col-10’) = 2.44 -33*r(sro); \]

Since the set \( sro \) was declared as a subset of the set \( row \), we can use \( sro \) as a controlling index in the assignment above to make the assignment only for the elements of \( sro \).

4.26.2.3.3 Restricting the Domain: Conditionals  Conditional assignments are discussed in detail in section Conditional Assignments in chapter Conditional Expressions, Assignments and Equations. For details on the types of logical conditions, see section Logical Conditions. Here we present a simple example to illustrate:

\[ a(row,col)$[a(row,col) >= 100] = INF; \]

This assignment has the following effect: all elements of the parameter \( a \) whose value was at least 100 are assigned the value INF, while all other elements of \( a \) remain unchanged.

4.26.2.3.4 Restricting the Domain: Tuples  Tuples or multi-dimensional sets are introduced in section Many-to-Many Mapping. In this simple example we show how they may be used to restrict the domain. The example builds on the first example in this section. We repeat the whole code here for clarity.

```
sets row / r-1*r-10 /
col / c-1*c-10 /
sro(row) / r-7*r-10 /
set tuple(row,col) /r-1.c-1, r-1.c-10, r-10.c-1, r-10.c-10/;
parameters r(row) /r-1*r-7 4, r-8*r-10 5/
c(col) /c-1*c-5 3, c-6*c-10 2/;
parameters a(row,col), b(row,row);
a(row,col) = 13.2 + r(row)*c(col); 
a(tuple(row,col)) = 7 + r(row)*c(col); 
a(tuple) = 0.25 * a(tuple); 
```

Note that we have introduced the new set \( tuple \). It is two-dimensional and contains just four elements. As before, the parameter \( a \) is first assigned values for all its 100 elements. We then change some of these values using the set \( tuple \) as domain. The values of the elements of the parameter \( a \) that are not elements of the set \( tuple \) remain unchanged.

For a more elaborate example involving tuples, see section Filtering Sets in Assignments.
4.26 Data Manipulations with Parameters

4.26.2.4 Issues with Controlling Indices

Attention

The number of controlling indices on the left of the = sign should be at least as large as the number of indices on the right. There should be no index on the right-hand side of the assignment that is not present on the left unless it is operated on by an indexed operator. For more on indexed operators, see section Indexed Operations.

Consider the following statement:

\[ a(\text{row},'\text{col-2}') = 22 - c(\text{col}) ; \]

GAMS will flag this statement as an error since \text{col} is an index on the right-hand side of the equation but not on the left.

Note that there would be no error here if \text{col} were a singleton set. Since there is only one element in a singleton set, the intent and behavior is well-defined even when \text{col} is not under control.

Attention

Each set is counted only once to determine the number of controlling indices. If the intent is for a set to appear independently more than once within the controlling domain, the second and subsequent occurrences of the set should be aliases of the original set, so that the number of controlling indices is equal to the number of indices. For details on aliases, see section The Alias Statement: Multiple Names for a Set.

Consider the following statement as an illustration:

\[ b(\text{row},\text{row}) = 7.7 - r(\text{row}) ; \]

This statement has only one controlling index, namely \text{row}. One element (on the diagonal of \text{b}) is assigned for each element of \text{row}, for a total of 10 assigned values. None of the off-diagonal elements of \text{b} will be changed!

If the intent is to assign values to each element of \text{b}, this can be done by introducing an alias \text{rowp} for \text{row} and using this alias in the second index position. There will then be two controlling indices and GAMS will make assignments over all 100 values of the full Cartesian product. The following example illustrates this method:

\[ \text{alias}(\text{row},\text{rowp}) ; \]
\[ b(\text{row},\text{rowp}) = 7.7 - (r(\text{row}) + r(\text{rowp}))/2 ; \]

4.26.3 Expressions

An expression is an arbitrarily complicated specification for a calculation, with parentheses nested as needed for clarity and intent. In this section the discussion of parameter assignments will continue by showing in more detail the expressions that may be used on the right of the = sign. We will cover standard arithmetic operations and indexed operations here, and functions and extended range arithmetic in the next two sections.

4.26.3.1 Standard Arithmetic Operations

The standard arithmetic symbols and operations and their order of precedence are given in Table 1. Note that 1 denotes the highest order of precedence and 3 denotes the lowest order of precedence. Parentheses can be used to override the default precedence order in the usual way. Operators (including exponentiation) on the same level are evaluated from left to right.
Table 1: Standard Arithmetic Operations

Note that the full GAMS operator precedence hierarchy also includes logical operators; it is given in section Mixed Logical Conditions. Note further that the symbols for addition, subtraction and multiplication have a different meaning if they are used in the context of sets. For details see sections Set Operations and Lag and Lead Operators.

Attention

The operation \( x^{**}y \) is equivalent to the function \( rPower(x,y) \) and is calculated internally as \( e^{y \times \log(x)} \). This operation is not defined if \( x \) is negative; an error will result in this case. If the possibility of negative values for \( x \) is to be admitted and the exponent is known to be an integer, then the function \( power(x,n) \) may be used.

Like many GAMS intrinsic functions, the operation \( x^{**}y \) is not defined for all possible input values. When evaluating nonlinear functions and operators GAMS guards against evaluating at or very near singular points and in such cases signals an error or returns an appropriate function value.

Note

As usual, operations within parentheses or brackets are evaluated before other numerical calculations, where the innermost parentheses are resolved first. Any of the pairs ( ), [ ] and { } are allowed.

Consider for example:

\[ x = 5 + 4*3^{**}2; \]

For clarity, this could have been written as follows:

\[ x = 5 + (4*3^{**}2) ; \]

In both cases the result is 41.

Note

It is often better to use parentheses than to rely on the order of precedence of operators, since this prevents errors and clarifies intentions.

Note that expressions may be freely continued over many lines: an end of line is permissible at any point where a blank may be used. Blanks may be used for readability around identifiers, parentheses and operation symbols. Blanks are not allowed within identifiers or numbers, and are significant inside the quotes used to delimit labels.

4.26.3.2 Indexed Operations

In addition to the simple operations in Table 1, GAMS also provides the following six indexed operations.
Table 2: Indexed Operations

These six operations are performed over one or more controlling indices. The syntax in GAMS for these operations is:

\[ \text{indexed_op( (index_list), expression);} \]

Indexed_op is one of the four keywords for indexed operations, index_list is the list of the controlling indices and expression is the expression to be evaluated. If there is only one controlling index, the parentheses around it may be removed. Consider the following simple example adapted from [ANDEAN]:

```
sets i plants / cartagena, callao, moron /
    m product / nitr-acid, sulf-acid, amm-sulf /;
parameter capacity(i,m) capacity in tons per day
    totcap(m) total capacity by process ;

totcap(m) = sum(i, capacity(i,m));
```

The index over which the summation is done, i, is separated from the reserved word `sum` by a left parenthesis and from the data term `capacity(i,m)` by a comma. The set `i` is called the controlling index for this operation. The scope of the control is the pair of parentheses `()` that start immediately after the sum. Note that using normal mathematical representation the last line could be written as: \( \text{totC}_m = \sum_i \text{C}_{im} \).

It is also possible to sum simultaneously over the domain of two or more sets as in the first assignment that follows. The second assignment demonstrates the use of a less trivial expression than an identifier within the indexed operation.

```
count = sum((i,j), a(i,j)) ;
emp = sum(t, l(t)*m(t)) ;
```

The equivalent mathematical forms are:

\[ \text{count} = \sum_i \sum_j A_{ij} \quad \text{and} \quad \text{emp} = \sum_t L_t M_t. \]

Note that the following alternative notation may be used for the first assignment above:
count = sum(i, sum(j, a(i,j)));

In the context of sets the sum operator may be used to compute the number of elements in a set and for projections. For details see section Projection and Aggregation of Sets.

The smin and smax operations are used to find the largest and smallest values over the domain of the index set or sets. The index for the smin and smax operators is specified in the same manner as in the index for the sum operator. In the following example we want to find the largest capacity:

\[ lrgunit = \text{smax}((i,m), \text{capacity}(i,m)) \]

Attention

The indexed operations smin and smax may be used in equation definitions only if the corresponding model is of type DNLP. For more on GAMS model types, see GAMS Model Types.

Note

- In the context of assignment statements, the attributes of variables and equations (e.g. \text{x.up}(i,j)) may be used in indexed operations just as scalars and parameters are used. For more on variable and equations attributes, see sections Variable Attributes and Equation Attributes respectively.
- In the context of equation definitions, scalars, parameters and variables may appear freely in indexed operations. For more on equation definitions, see section Defining Equations.

Sometimes it is necessary to restrict the domain of indexed operations. This may be done with the same techniques as in indexed assignments, see section Restricting the Domain in Assignments for details. See also section Conditional Indexed Operations for more details on conditions in the context of indexed operations.

We now turn to two additional capabilities that are available in GAMS to add power and flexibility to expression calculations: functions and extended range arithmetic.

### 4.26.4 Functions

Functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or intrinsic functions. GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and quite sophisticated or specialized functions. There is a trade-off between satisfying these requests and avoiding a complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, see subsection Extrinsic Functions below.
4.26.4 Intrinsic Functions

GAMS provides many functions, ranging from commonly used standard functions like exponentiation, logarithms, and trigonometric functions to utility functions for controlling and querying the running GAMS job or process. The complete list of available functions is given in the following tables: Mathematical Functions, String Manipulation Functions, Logical Functions, Time and Calendar Functions, and GAMS Utility and Performance Functions. For details specific to using these functions in equations, see the section on Expressions in Equation Definitions.

Some of the tables that follow contain an endogenous classification column "End. Classif." that specifies in which models the function may legally appear. In order of least to most restrictive, the choices are DNLP, NLP, any, none. See section Classification of Models for details on model types in GAMS. Note well: functions classified as any are only permitted with exogenous (constant) arguments.

Functions are typically used in assignment statements and equations. In these cases, they are only evaluated at execution time. Some functions can also be used at compile time, e.g. to compute the factorial or square root of a scalar. Some of the tables below contain a column "Compile Time" indicating which functions can be used at compile time.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogenous variables, see section Functions in Equation Definitions. Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.

4.26.4.1.1 Mathematical Functions Mathematical functions may be used as expressions in assignment statements and in equation definitions. We start with some simple examples to illustrate. A list with all mathematical functions available in GAMS is given in Table 3.

Exp(x)

\[ a = \exp(t); \]
\[ b = \exp(t+2); \]

The GAMS function \( \exp(x) \) returns the exponential \( e^x \) of its argument. The assignments above set \( a = e^t \) and \( b = e^{t+2} \) respectively.

Log(x)

\[ z(j) = \log(y(j)); \]

The GAMS function \( \log(x) \) returns the natural logarithm of its argument. The assignment above evaluates the logarithm once for each element of the controlling domain \( j \).

Max(x1,x2,x3,...)

\[ x = \max(y+2, t, t\cdot t); \]

The function \( \max \) returns the maximum of a set of expressions or terms. In the assignment above, \( x \) will be assigned the maximum of \( y + 2, t, \) and \( t \cdot t \).

Round(x[,DECPL])

The function \( \text{round} \) rounds its argument \( x \) to the specified number \( \text{DECPL} \) of places, where positive values of \( \text{DECPL} \) indicating rounding to the right of the decimal point. If the argument \( \text{DECPL} \) is not specified, it defaults to zero, and the function rounds its argument to the nearest integer value. For example,
In the first assignment \( q \) is rounded to the nearest integer value, while in the second \( q \) is rounded to the number of decimals specified by \( d \). The result of the third assignment is 12.43, while the final assignment results in a value of 520.

Table 3: Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td><strong>Absolute value</strong> of the argument ( x ).</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>arccos(x)</td>
<td><strong>Inverse cosine</strong> of the argument ( x ), where ( x ) is a real number between -1 and 1 and the output is in radians, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>arcsin(x)</td>
<td><strong>Inverse sine</strong> of the argument ( x ), where ( x ) is a real number between -1 and 1 and the output is in radians, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>arctan(x)</td>
<td><strong>Inverse tangent</strong> of the argument ( x ), where ( x ) is a real number and the output is in radians, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>arctan2(y,x)</td>
<td><strong>Four-quadrant arctan function</strong> yielding ( \arctan\left(\frac{y}{x}\right) ) which is the angle the vector ( (x, y) ) makes with ( (1,0) ) in radians.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>beta(x,y)</td>
<td><strong>Beta function</strong>: ( B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} ), see MathWorld</td>
<td>DNLP</td>
<td>no</td>
</tr>
<tr>
<td>betaReg(x,y,z)</td>
<td><strong>Regularized beta function</strong>, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>binomial(n,k)</td>
<td>(Generalized) <strong>Binomial coefficient</strong> for ( n &gt; -1, -1 &lt; k &lt; n + 1 ).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>ceil(x)</td>
<td><strong>Ceiling</strong>: returns the smallest integer greater than or equal to ( x ).</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>centropy(x,y[,Z])</td>
<td><strong>Cross-entropy</strong>: ( x \cdot \ln\left(\frac{x+Z}{y+Z}\right) ), where ( Z \geq 0 ). Default ( Z ): ( 1e-20 ).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>cos(x)</td>
<td><strong>Cosine</strong> of the argument ( x ), where ( x ) must be in radians, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>cosh(x)</td>
<td><strong>Hyperbolic Cosine</strong> of ( x ), where ( x ) must be in radians, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>cvPower(X,Y)</td>
<td><strong>Real power</strong>: returns ( X^Y ) for ( X \geq 0 ).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>div(dividend,divisor)</td>
<td>Returns ( \frac{\text{dividend}}{\text{divisor}} ), undefined for ( \text{divisor} = 0 ).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>div0(dividend,divisor)</td>
<td>Returns ( \frac{\text{dividend}}{\text{divisor}} ), returns 10^{299} for ( \text{divisor} = 0 ).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>End. Classif.</td>
<td>Compile Time</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>eDist(x1[,x2,x3,...])</td>
<td>Euclidean or L-2 Norm: $\sqrt{x_1^2 + x_2^2 + \ldots}$</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>entropy(x)</td>
<td>Entropy: $-x \cdot \ln(x)$</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>erf(x)</td>
<td>Integral of the standard normal distribution from negative infinity to $x$: $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt.$</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>execSeed</td>
<td>Reads or resets the seed for the random number generator (RNG). Note that the state of the RNG cannot typically be captured in one seed value; in such cases &quot;reading&quot; the seed involves harvesting a seed from the RNG, resetting the RNG with this seed, and returning the seed.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential of $x$: $e^x$, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>fact(N)</td>
<td>Factorial of $N$, where $N \geq 0$ is an integer.</td>
<td>any</td>
<td>yes</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Floor: greatest integer less than or equal to $x$.</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>frac(x)</td>
<td>Returns the fractional part of $x$, s.t. $x = \text{trunc}(x) + \text{frac}(x)$</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>gamma(x)</td>
<td>Gamma function: $\Gamma(x) = \int_{0}^{\infty} t^{x-1}e^{-t} dt,$ see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>gammaReg(x,a)</td>
<td>Lower Incomplete Regularized Gamma function, see GammaRegularized[a,0,x] in MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>log(x)</td>
<td>Natural logarithm: logarithm base $e$, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>logBeta(x,y)</td>
<td>Log Beta function: $\log(B(x,y))$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>logGamma(x)</td>
<td>Log Gamma function as discussed in MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>logit(x)</td>
<td>Logit Transformation: $\log\left(\frac{x}{1-x}\right)$, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>log10(x)</td>
<td>Common logarithm: logarithm base 10, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>log2(x)</td>
<td>Binary logarithm: logarithm base 2, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>max(x1,x2,x3,...)</td>
<td>Maximum value of the arguments, where the number of arguments may vary.</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>min(x1,x2,x3,...)</td>
<td>Minimum value of the arguments, where the number of arguments may vary.</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>End. Classif.</td>
<td>Compile Time</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>mod(x,y)</td>
<td>Remainder of x divided by y.</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>ncpCM(x,y,Z)</td>
<td>Chen-Mangasarian smoothing: (x - Z \cdot \ln(1 + e^{\frac{x}{y}})).</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>ncpF(x,y[Z])</td>
<td>Fisher-Burmeister smoothing: (\sqrt{(x^2 + y^2 + 2 \cdot Z)} - x - y). Z \geq 0. Default Z = 0.</td>
<td>NLP</td>
<td>no</td>
</tr>
</tbody>
</table>
| ncpVUpow(r,s[MU])   | NCP Veelken-Ulbrich: smoothed min(r,s) \[
\begin{cases} 
(r+s-|t|) & \text{if } |t| \geq \mu \geq 0 \\
(r+s-\frac{t}{2} \cdot (-\frac{1}{2} t^2+6t^2+3)) & \text{otherwise} 
\end{cases}
\]
where \(t = r - s\). Default \(MU = 0\). | NLP           | no           |
| ncpVUsin(r,s[MU])   | NCP Veelken-Ulbrich: smoothed min(r,s) \[
\begin{cases} 
(r+s-|t|) & \text{if } |t| \geq \mu \geq 0 \\
(r+s-(\frac{2\pi}{\mu} \sin(\frac{\pi}{2}\mu)) \cdot |t|) & \text{otherwise} 
\end{cases}
\]
where \(t = r - s\). Default \(MU = 0\). | NLP           | no           |
<p>| normal(MEAN,STDDEV) | Generate a random number from the normal distribution with mean MEAN and standard deviation STDDEV, see <a href="https://mathworld.wolfram.com/NormalDistribution.html">MathWorld</a> | none          | no           |
| pi                  | Value of (\pi = 3.141593...)                                            | any           | yes          |
| poly(x,A0,A1,A2[A3,...]) | Returns (p(x)), where the polynomial (p(x) = A_0 + A_1 x + A_2 x^2 + A_3 x^3 + ... + A_{20} x^{20}). By default (A_3,...,A_{20} = 0). | NLP           | no           |
| power(x,Y)          | Returns (x^Y), where Y must be an integer.                               | NLP           | no           |
| randBinomial(N,P)   | Generate a random number from the binomial distribution, where N is the number of trials and P the probability of success for each trial, see <a href="https://mathworld.wolfram.com/BinomialDistribution.html">MathWorld</a> | none          | no           |
| randLinear(LOW,SLOPE,HIGH) | Generate a random number between LOW and HIGH with linear distribution, SLOPE must be greater than (2 \cdot \frac{HIGH-LOW}{LOW^2}) | none          | no           |
| randTriangle(LOW,MID,HIGH) | Generate a random number between LOW and HIGH with triangular distribution, MID is the most probable number, see <a href="https://mathworld.wolfram.com/TriangularDistribution.html">MathWorld</a> | none          | no           |
| round(x[,DECPL])    | Round x to DECPL decimal places. Default DECPL=0 | DNLP          | yes          |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rPower(x,y)</td>
<td>Returns $x^y$ for $x &gt; 0$ and also for $x = 0$ and restricted values of $y$. This function is equivalent to the operation 'x**y', see Standard Arithmetic Operations.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sigmoid(x)</td>
<td>Sigmoid: $\frac{1}{1+e^{-x}}$, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sign(x)</td>
<td>Sign of $x$: returns 1 if $x &gt; 0$, -1 if $x &lt; 0$, and 0 if $x = 0$.</td>
<td>DNLP</td>
<td>yes</td>
</tr>
<tr>
<td>signPower(x,Y)</td>
<td>Signed power: for $Y &gt; 0$, returns $x^Y$ if $x \geq 0$ and $-1 \cdot</td>
<td>x</td>
<td>^Y$ if $x &lt; 0$.</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Sine of the argument $x$, where $x$ must be in radians, see MathWorld</td>
<td>NLP</td>
<td>yes</td>
</tr>
<tr>
<td>sinh(x)</td>
<td>Hyperbolic sine of $x$, where $x$ must be in radians, see MathWorld</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>slexp(x[,S])</td>
<td>Smooth (linear) exponential: [ \begin{cases} e^x &amp; \text{if } x \leq S \ e^x \cdot (1 + (x - S)) &amp; \text{otherwise} \end{cases} ] where $S \leq 150$, default $S = 150$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sllog10(x[,S])</td>
<td>Smooth (linear) logarithm base 10: [ \begin{cases} \log_{10}(x) &amp; \text{if } x \geq S \ \frac{1}{\ln 10} \cdot (\ln S + \frac{x-S}{S^2}) &amp; \text{otherwise} \end{cases} ] where $S \geq -150$, default $S = -150$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>slrec(x[,S])</td>
<td>Smooth (linear) reciprocal: [ \begin{cases} \frac{1}{x} &amp; \text{if } x \geq S \ \frac{1}{S} - \frac{x-S}{S} &amp; \text{otherwise} \end{cases} ] where $S \geq 10^{-10}$, default $S = 10^{-10}$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sqexp(x[,S])</td>
<td>Smooth (quadratic) exponential: [ \begin{cases} e^x &amp; \text{if } x \leq S \ e^x \cdot (1 + (x - S) + \frac{(x-S)^2}{2}) &amp; \text{otherwise} \end{cases} ] where $S \leq 150$, default $S = 150$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sqlog10(x[,S])</td>
<td>Smooth (quadratic) logarithm base 10: [ \begin{cases} \log_{10}(x) &amp; \text{if } x \geq S \ \frac{1}{\ln 10} \cdot (\ln S + \frac{x-S}{S} - \frac{(x-S)^2}{2S^2}) &amp; \text{otherwise} \end{cases} ] where $S \geq -150$, default $S = -150$.</td>
<td>NLP</td>
<td>no</td>
</tr>
<tr>
<td>sqr(x)</td>
<td>Square of argument $x$.</td>
<td>NLP</td>
<td>yes</td>
</tr>
</tbody>
</table>
Function | Description | End. Classif. | Compile Time
--- | --- | --- | ---
sqrec(x[,S]) | Smooth (quadratic) reciprocal: result = \[
\begin{align*}
\frac{1}{x} & \quad \text{if } x \geq S \\
\frac{1}{S} - \frac{x-S}{S^2} + \frac{(x-S)^2}{S^3} & \quad \text{otherwise}
\end{align*}
\] where \( S \geq 10^{-10} \), default \( S = 10^{-10} \). | NLP | no
sqrt(x) | Square root of \( x \), see MathWorld | NLP | yes
tan(x) | Tangent of the argument \( x \), where \( x \) must be in radians, see MathWorld | NLP | yes
tanh(x) | Hyperbolic tangent of \( x \), where \( x \) must be in radians, see MathWorld | NLP | no
trunc(x) | Truncation: returns the integer part of \( x \), truncating towards zero. | DNLP | yes
uniform(LOW,HIGH) | Generates a random number from the uniform distribution between \( LOW \) and \( HIGH \), see MathWorld | none | no
uniformInt(LOW,HIGH) | Generates an integer random number from the discrete uniform distribution whose outcomes are the integers between \( LOW \) and \( HIGH \), inclusive, see MathWorld | none | no
vcPower(x,Y) | Returns \( x^Y \) for \( x \geq 0 \). | NLP | no

### 4.26.4.1.2 String Manipulation Functions
GAMS provides some string manipulation capability by extending the `card` and `ord` functions to work on strings as well as sets. In Table 4 the extended behavior is described. In this context, the functions take strings and places as arguments, and the numeric places argument must be a constant. This behavior only applies to execution-time usage of these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>card(STRING)</strong></td>
<td>Returns the number of characters in the string.</td>
</tr>
<tr>
<td><strong>ord(STRING[,PLACE])</strong></td>
<td>Returns the ASCII code number of a character in a position in a string. The optional place entry defaults to 1 if omitted and identifies the character position within the string to be examined (1 for the first, 2 for the second, etc.).</td>
</tr>
</tbody>
</table>

There are four types or sources of strings in this context. A string may be a string literal, i.e. a concatenation of letters and blanks as in "drink it" in the example that follows. It may be the symbol text (aka the explanatory text) associated with any symbol. Or it may be the labels or text associated with the elements of a set. The following table gives an overview:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>card(STRING)</strong></td>
<td>Returns the number of characters in the string.</td>
</tr>
<tr>
<td><strong>ord(STRING[,PLACE])</strong></td>
<td>Returns the ASCII code number of a character in a position in a string. The optional place entry defaults to 1 if omitted and identifies the character position within the string to be examined (1 for the first, 2 for the second, etc.).</td>
</tr>
</tbody>
</table>
4.26 Data Manipulations with Parameters

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>characters</td>
<td>A concatenation of characters and blanks: all legal GAMS characters are</td>
<td>The characters must be surrounded by double or single quotes.</td>
</tr>
<tr>
<td></td>
<td>allowed, see section Characters for details.</td>
<td></td>
</tr>
<tr>
<td>symbol_name.ts</td>
<td>The string is the explanatory text associated with a symbol name.</td>
<td>If the explanatory text is missing, the value of card is 0.</td>
</tr>
<tr>
<td>set_name.tl</td>
<td>The string is the label for a set element.</td>
<td>This string type may be used only when set_name is part of the controlling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>domain.</td>
</tr>
<tr>
<td>set_name.te</td>
<td>The string is the explanatory text associated with a set element.</td>
<td>This string type may be used only when set_name is part of the controlling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>domain. If the explanatory text is missing, the value of card is 0.</td>
</tr>
</tbody>
</table>

The string types are illustrated in the following example. Note that the directive in the first line activates the end-of-line comment option, see eolCom.

```
$oneolcom
variable z 'any symbol can have explanatory text';
set teas "teas available to order" / black "English Breakfast", green, peppermint /;
scalar p;
  p = card("drink it"); !! result: p=8
  p = card(z.ts);     !! result: p=36
  p = card(teas.ts);  !! result: p=23
loop{teas,
  p = card(teas.tl)   !! length of set element label from the set "teas" : "teas" is the controlling set
  display "length of set element label", p;
  p = card(teas.te)   !! length of set element explanatory text: "teas" is the controlling set
  display "length of set element explanatory text", p;
};
```

Note that the strings teas.tl and teas.te are used in the context of a loop statement (see section The Loop Statement). This is a typical usage pattern.

4.26.4.1.3 Logical Functions Logical functions may be used as expressions in assignment statements as in the following example.

```
x = ifthen(tt=2, 3, 4+y);
```

Here \( x = 3 \) if \( tt = 2 \), otherwise \( x = 4 + y \).

The logical functions available in GAMS are given in Table 6. Note that logical functions may also be used in conditions and logic equations, see sections Logical Conditions and Logic Equations respectively. Most of the logical functions can also be indicated using the familiar operator notation, e.g. \((x \text{ and } y), (x \geq y), \text{ etc.}\). In such cases, the operator notation is allowed at compile time. The last column in Table 6 indicates if a logical function can be used at compile time (yes) or if only its alternative notation is available (alt). Further, note that the inputs and outputs of these functions are often logical/Boolean values, so GAMS does implicit conversions as necessary. As expected, false becomes 0 and true becomes 1 when converting logical values to numeric, and 0 becomes false and nonzero becomes true when numeric values are converted to logicals. For details on behavior when the inputs are special values, see sections Extended Range Arithmetic and Acronym Usage, but note that EPS, +INF, -INF, and acronyms become true when converted to logicals.

**Table 6: Logical Functions**
<table>
<thead>
<tr>
<th>Function</th>
<th>Alternative Notation 1</th>
<th>Alternative Notation 2</th>
<th>Description</th>
<th>Return Values</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_and(x,y)</td>
<td>x and y</td>
<td>Boolean AND</td>
<td>Returns true iff both x and y are true</td>
<td>DNLP</td>
<td>alt</td>
<td></td>
</tr>
<tr>
<td>bool_eqv(x,y)</td>
<td>x eqv y</td>
<td>x &lt;=&gt; y</td>
<td>Boolean equivalence</td>
<td>Returns false iff exactly one argument is false</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>bool_imp(x,y)</td>
<td>x imp y</td>
<td>x -&gt; y</td>
<td>Boolean implication</td>
<td>Returns true iff x is false or y is true</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>bool_not(x)</td>
<td>not x</td>
<td>Boolean NOT</td>
<td>Returns true iff x is false</td>
<td>DNLP</td>
<td>alt</td>
<td></td>
</tr>
<tr>
<td>bool_or(x,y)</td>
<td>x or y</td>
<td>Boolean OR</td>
<td>Returns true iff x is true or y is true</td>
<td>DNLP</td>
<td>alt</td>
<td></td>
</tr>
<tr>
<td>bool_xor(x,y)</td>
<td>x xor y</td>
<td>Boolean XOR</td>
<td>Returns true iff exactly one argument is false</td>
<td>DNLP</td>
<td>alt</td>
<td></td>
</tr>
<tr>
<td>ifThen(cond,iftrue,else)</td>
<td></td>
<td>Condition</td>
<td>If the logical condition (first argument) is true, the function returns iftrue, else it returns else. See example above.</td>
<td>DNLP</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>rel_eq(x,y)</td>
<td>x eq y</td>
<td>x = y</td>
<td>Numeric Relation: Equal</td>
<td>Returns true iff x = y</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>rel_ge(x,y)</td>
<td>x ge y</td>
<td>x &gt;= y</td>
<td>Numeric Relation: Greater Equal</td>
<td>Returns true iff x ≥ y</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>rel_gt(x,y)</td>
<td>x gt y</td>
<td>x &gt; y</td>
<td>Numeric Relation: Greater Than</td>
<td>Returns true iff x &gt; y</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>rel_le(x,y)</td>
<td>x le y</td>
<td>x &lt;= y</td>
<td>Numeric Relation: Less Equal</td>
<td>Returns true iff x ≤ y</td>
<td>DNLP</td>
<td>alt</td>
</tr>
<tr>
<td>rel_lt(x,y)</td>
<td>x lt y</td>
<td>x &lt; y</td>
<td>Numeric Relation: Less Than</td>
<td>Returns true iff x &lt; y</td>
<td>DNLP</td>
<td>alt</td>
</tr>
</tbody>
</table>
4.26 Data Manipulations with Parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Alternative Notation 1</th>
<th>Alternative Notation 2</th>
<th>Description</th>
<th>Return Values</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel_ne(x,y)</td>
<td>x ne y</td>
<td>x &lt;&gt; y</td>
<td>Numeric Relation: Not Equal</td>
<td>Returns true iff ( x \neq y )</td>
<td>DNLP</td>
<td>alt</td>
</tr>
</tbody>
</table>

4.26.4.1.4 Time and Calendar Functions GAMS offers several functions that relate to time and dates. The fundamental measurement of time in GAMS is the serial day number beginning with January 1, 1900. This serial day number is a real number whose integer part contains a unique number for each day and whose fractional part contains information about hours, minutes, and seconds. We can think of the serial day number as being a date.time pair. The day information extracted from serial day numbers is based on the Gregorian calendar.

Note

In all functions given in Table 7, serial day 1 is January 1, 1900.

All of the functions in Table 7 can be used at compile time.

Table 7: Time and Calendar Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>gday(SDAY)</td>
<td>Returns Gregorian day from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gdow(SDAY)</td>
<td>Returns Gregorian day of week from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>ghour(SDAY)</td>
<td>Returns Gregorian hour of day from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gleap(SDAY)</td>
<td>Returns 1 if the year that corresponds to a serial day number date.time, is a leap year, else returns 0.</td>
<td>any</td>
</tr>
<tr>
<td>gmillisec(SDAY)</td>
<td>Returns Gregorian milli second from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gminute(SDAY)</td>
<td>Returns Gregorian minute of hour from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gmonth(SDAY)</td>
<td>Returns Gregorian month from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gsecond(SDAY)</td>
<td>Returns Gregorian second of minute from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>gyear(SDAY)</td>
<td>Returns Gregorian year from a serial day number date.time.</td>
<td>any</td>
</tr>
<tr>
<td>jdate(YEAR,MONTH,DAY)</td>
<td>Returns a serial day number.</td>
<td>any</td>
</tr>
<tr>
<td>jnow</td>
<td>Returns the current time as a serial day number.</td>
<td>none</td>
</tr>
<tr>
<td>jstart</td>
<td>Returns the time of the start of the GAMS job as a serial day number.</td>
<td>none</td>
</tr>
<tr>
<td>jtime(HOUR,MIN,SEC)</td>
<td>Returns fraction of a day that corresponds to hour, minute and second.</td>
<td>any</td>
</tr>
</tbody>
</table>

4.26.4.1.5 GAMS Utility and Performance Functions GAMS provides several functions that may be used to get (and in some cases set) GAMS system information, for example:

```
scalar o;
```
```
o = heapLimit;
heapLimit = 1024;
```

### Table 8: GAMS Utility and Performance Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>embeddedHandle</code></td>
<td>Returns the handle for the last embedded code section executed, see section Syntax in chapter Embedded Code Facility for details.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>errorLevel</code></td>
<td>Return code of the most recently used command.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td><code>execError</code></td>
<td>Get or set the number of execution errors.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>gamsRelease</code></td>
<td>Returns the release number of the running GAMS system, for example 24.7.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td><code>gamsVersion</code></td>
<td>Returns the version number of the running GAMS system, for example 247.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td><code>handleCollect</code></td>
<td>Tests if the solve of the model instance identified by the calling argument HANDLE is done: if so, it loads the solution into GAMS. For details, see Table 1 in section Grid Computing.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>handleDelete</code></td>
<td>Deletes the model instance identified by HANDLE. For details, see Table 1 in section Grid Computing.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>handleStatus</code></td>
<td>Tests if the solve of the model instance identified by HANDLE is done: if so, it loads the solution into a GDX file. For details, see Table 1 in section Grid Computing.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>handleSubmit</code></td>
<td>Resubmits the model instance identified by HANDLE for solution. For details, see Table 1 in section Grid Computing.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>heapFree</code></td>
<td>Get the amount of free memory in the heap in MB, i.e. memory allocated to the process and available for future use by GAMS.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>heapLimit</code></td>
<td>Get or set the current heap limit (maximum allowable dynamic memory usage) in Mb.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>heapSize</code></td>
<td>Get the current heap size in Mb.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td><code>jobHandle</code></td>
<td>Returns the process ID (PID) of the last job started.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>End. Classif.</td>
<td>Compile Time</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>jobKill(PID)</td>
<td>Sends a kill signal to the job with process ID PID. The return value is 1 if this was successful, 0 otherwise.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>jobStatus(PID)</td>
<td>Get the status of the job with process ID PID. Possible return values are: 0: error (input is not a valid PID or access is denied) 1: process is still running 2: process is finished with return code which could be accessed by errorlevel 3: process not running anymore or was never running, no return code available</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>jobTerminate(PID)</td>
<td>Sends an interrupt signal to the job with process ID PID. The return value is 1 if this was successful, 0 otherwise.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>licenseLevel</td>
<td>Get an indicator for the type of license: 0: demo license, limited to small models 1: full unlimited developer license 2: run time license, no new variables or equations can be introduced besides those inherited from a work file 3: application license, only works with a specific work file which is locked to the license file 5: community license, limited to somewhat bigger models compared to demo license.</td>
<td>any</td>
<td>no</td>
</tr>
<tr>
<td>licenseStatus</td>
<td>Returns non-zero if a license error has occurred.</td>
<td>any</td>
<td>no</td>
</tr>
<tr>
<td>mapVal(x)</td>
<td>Returns an integer value that indicates what special value (if any) is stored in the input x. Possible results: 0: x is not a special value 4: x is UNDF (undefined) 5: x is NA (not available) 6: x is INF (∞) 7: x is -INF (-∞) 8: x is EPS &gt;8: x is an acronym</td>
<td>any</td>
<td>no</td>
</tr>
<tr>
<td>maxExecError</td>
<td>Get or set the maximum number of execution errors. See also maxExecError.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>numCores</td>
<td>Get the number of logical CPU cores in the system</td>
<td>any</td>
<td>yes</td>
</tr>
</tbody>
</table>
### 4.26.4.2 Extrinsic Functions

Using the GAMS Function Library Facility, functions may be imported from an external library into a GAMS model. Apart from the import syntax, the imported functions may be used in the same way as intrinsic functions. In particular, they may be used in equation definitions. Some function libraries are included with the standard GAMS software distribution, but GAMS users can also create their own libraries using an open programming interface. The GAMS Test Library instances [TRILIB01], [TRILIB02], [TRILIB03], and [CPPLIB00] are simple examples (in the programming languages C, Delphi, Fortran and C++, respectively) that come with every GAMS system.

For details on using and creating extrinsic function libraries, and on the extrinsic function libraries that are provided with GAMS, see the chapter on Extrinsic Functions.

### 4.26.4.3 Function Suffixes

Up to this point, this section has described and discussed functions without mentioning their derivatives. These derivatives are very important, though: most of the nonlinear solvers integrated with GAMS will require first derivatives to solve models, and many will also use or require second derivatives as well. It is sometimes useful (e.g. when testing an extrinsic function) to evaluate the derivatives of GAMS functions instead of the functions themselves. This can be done via function suffixes.

Function suffixes can specify the evaluation of a gradient (i.e. first derivative), a Hessian (i.e. second derivative), a minimum or maximum value of the function over a given range, or the minimum or maximum value of the gradient over a given range. A full list of function suffixes is given in Table 9. For functions whose arguments are constant, the derivatives are zero, so typically `func` is a mathematical function listed in Table 3 above. Note that function suffixes are not defined for functions without arguments (for example, `pi`), so specifying something like `pi.grad` results in a compilation error.

#### Table 9: Function Suffixes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
<th>Compile Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>platformCode</td>
<td>Returns an integer indicating the current platform. Possible return values can be seen here.</td>
<td>any</td>
<td>yes</td>
</tr>
<tr>
<td>readyCollect(HANDLES[,MAXWAIT])</td>
<td>Waits until a model solution is ready to be collected. For details, see Table 1 in section Grid Computing.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>sleep(SEC)</td>
<td>Pause or sleep execution for SEC seconds.</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>timeClose</td>
<td>Returns the accumulated closedown time, i.e. the time GAMS uses to save its state to disk prior to a solve.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>timeComp</td>
<td>Returns the compilation time in seconds.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>timeElapsed</td>
<td>Returns the elapsed time since the start of a GAMS run in seconds.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>timeExec</td>
<td>Returns the execution time in seconds.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>timeStart</td>
<td>Returns the accumulated startup time, i.e. the time GAMS uses to restore its state from disk after a solve.</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>Function Suffix</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.value(x)</code></td>
<td>Value of the function <code>func</code> at <code>x</code>, equals <code>func(x)</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.grad(x)</code></td>
<td>Value of the gradient of the function <code>func</code> at <code>x</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.gradn(x)</code></td>
<td>Value of the gradient of the function <code>func</code> at <code>x</code>, computed numerically.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.hess(x)</code></td>
<td>Value of the Hessian of the function <code>func</code> at <code>x</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.hessn(x)</code></td>
<td>Value of the Hessian of the function <code>func</code> at <code>x</code>, computed numerically.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.low(x1:x2)</code></td>
<td>Lower bound of the function <code>func(x)</code> on the interval <code>[x1,x2]</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.high(x1:x2)</code></td>
<td>Upper bound of the function <code>func(x)</code> on the interval <code>[x1,x2]</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.gradL(x1:x2)</code></td>
<td>Lower bound of the gradient of the function <code>func(x)</code> on the interval <code>[x1,x2]</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>func.gradH(x1:x2)</code></td>
<td>Upper bound of the gradient of the function <code>func(x)</code> on the interval <code>[x1,x2]</code>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider the following simple example:

```plaintext
scalars g, l, h, gl, gh;

  g = sin.grad(0);
  l = sin.low(pi/3:pi/2);
  h = sin.high(pi/3:pi/2);
  gl = sin.gradL(pi/3:pi/2);
  gh = sin.gradH(pi/3:pi/2);

display g, l, h, gl, gh;
```

For univariate functions like sine or cosine, there is no need to specify a variable index when evaluating derivatives. For multivariate functions, the default is to take partial derivatives w.r.t. the first variable. To specify other variables, the colon syntax in the example below is used.

```plaintext
scalars
  x / 1 /
  ylo / -1 /
  yup / 2 /
  e0, e1, elo, ehi
  e_1, e_2
  e_11, e_22, e_21
;

  e0 = edist(x,ylo);
  e1 = edist(x,yup);
  elo = edist.low (x,ylo:x,yup);
  ehi = edist.high(x,ylo:x,yup);

  e_1 = edist.grad(x,yup);
  * e_1 = edist.grad(1:x,yup); same as above
  e_2 = edist.grad(2:x,yup);

  e_11 = edist.hess(x,yup);
  * e_11 = edist.hess(1:1:x,yup); same as above
  e_22 = edist.hess(2:2:x,yup);
  e_21 = edist.hess(2:1:x,yup);

display x, ylo, yup, e0, e1, elo, ehi, e_1, e_2, e_11, e_22, e_21;
```
For more examples, see model \([\text{FUNCS4}]\) in the GAMS Test Library.

**Note**

- The function suffixes **value**, **grad**, **gradn**, **hess** and **hessn** are also defined for extrinsic functions. When implementing an extrinsic function, be aware that missing derivatives will be computed numerically: see model \([\text{TRILIB01}]\) for an example.

To compute derivatives numerically, GAMS uses finite difference approximations. This computation can be controlled with two options: the **FDOpt** option controls which variant of the finite difference method is used, while the **FDDelta** option controls the step size.

### 4.26.5 Extended Range Arithmetic and Error Handling

GAMS uses an *extended range* arithmetic to handle missing data, the results of undefined operations, infinite values, and zeros that are stored explicitly. The *special values* used in this arithmetic are listed and described in Table 10 below, along with the value of the **mapVal** function that corresponds to these values. We can think of special values as any value for which **mapVal** does not return 0.

<table>
<thead>
<tr>
<th>Special value</th>
<th>Description</th>
<th>mapVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>INF</td>
<td>Plus infinity. Similar to IEEE plus infinity. Behaves in the expected way in computations, so that e.g. min(x,INF) = x unless x is also special.</td>
<td>6</td>
</tr>
<tr>
<td>-INF</td>
<td>Minus infinity. Similar to IEEE minus infinity. Behaves in the expected way in computations, so that e.g. max(x,INF) = x unless x is also special.</td>
<td>7</td>
</tr>
<tr>
<td>NA</td>
<td>Not available - used to indicate missing data. It is a sticky value: e.g. max(x,NA) = NA even for x = INF</td>
<td>5</td>
</tr>
<tr>
<td>UNDF</td>
<td>Undefined - indicates the result of an undefined or illegal operation. Similar to IEEE NaN. A user cannot directly set a value to UNDF unless the dollar control option onUNDF is active. For details, see the chapter on Dollar Control Options.</td>
<td>4</td>
</tr>
<tr>
<td>EPS</td>
<td>A stored zero value. If the dollar control option onEPS is active, zeros in a parameter or table statement are treated as EPS. For details on dollar control options, see chapter Dollar Control Options.</td>
<td>8</td>
</tr>
</tbody>
</table>

**Table 10:** Special Values for Extended Range Arithmetic

**Attention**

Avoid creating or using numbers with very large (1.0e299 or above) or very small (1.0e-299 or below) magnitudes. Large numbers may be treated by GAMS as undefined (UNDF) or other special values, leading to unpredictable and unusable results. Always use INF (or -INF) explicitly for arbitrarily large (or small) numbers.

GAMS has defined the results of all arithmetic operations and all function evaluations that use these special values. The behavior is designed to both maximize utility and minimize surprise, and is illustrated in the library model \([\text{CRAZY}]\). For example: 1+INF evaluates to INF, 1-EPS to 1, NA * 2 to NA, and EPS+INF to UNDF.

The following table shows a selection of results for exponentiation and division for a variety of input parameters including NA and INF.
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a**b</th>
<th>power(a,b)</th>
<th>a/b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>-2</td>
<td>2</td>
<td>UNDF</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>2.1</td>
<td>4.28</td>
<td>UNDF</td>
<td>.952</td>
</tr>
<tr>
<td>NA</td>
<td>2.5</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>UNDF</td>
</tr>
<tr>
<td>INF</td>
<td>2</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
</tr>
<tr>
<td>2</td>
<td>INF</td>
<td>UNDF</td>
<td>UNDF</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 11: Extended Range Arithmetic in Exponentiation and Division

Note that most extended range identifiers may be used in assignment statements, as illustrated below.

\[
a(\text{row},'\text{col}-3') = \text{NA};
\]
\[
a(\text{row},'\text{col}-4') = \text{INF};
\]
\[
a(\text{row},'\text{col}-5') = -\text{INF};
\]

The values most often used in assignments are \text{NA} in incomplete tables and \text{INF} for variable bounds.

The special value \text{EPS} is numerically equal to zero, so when used in the context of numerical comparisons, it behaves as zero. For example, the logical expressions \(x > 0\) and \(x > \text{EPS}\) have the same result. However, \text{EPS} is a stored zero, so setting a parameter to zero (e.g. \(a(\text{row},\text{col})=0\)) results in no values being stored for \(a\), while setting a parameter to \text{EPS} (e.g. \(a(\text{row},\text{col})=\text{EPS}\)) results in the value \text{EPS} being stored for every \((\text{row},\text{col})\) tuple. This is sometimes useful, especially if \(a\) is used later in loops (see Programming Flow Control Features) or dollar conditions (see Conditional Expressions, Assignments and Equations). For example, consider the statement

\[
a(\text{row},\text{col})\$[a(\text{row},\text{col})] = \text{INF} ;
\]

In the case where \(a\) was originally assigned to be zero, the dollar-condition is always false so no infinities are assigned. If \(a\) was originally assigned the value \text{EPS}, the dollar-condition is always true and every tuple in \(a\) will be set to infinity.

When an attempted arithmetic operation is illegal or has undefined results because of the value of arguments (division by zero is the normal example), an error is reported and the result is set to undefined (\text{UNDF}). The error is marked on the output file with a row of four asterisks \(\ast\ast\ast\ast\) making this sequence a good search target in the editor. GAMS will also report the line number of the offending statement and give as much detail as possible about the cause. From there on, the resulting \text{UNDF} is treated as a proper data value and does not trigger additional error messages. For more on error messages, see chapter GAMS Output.

Note

GAMS will not solve a model if an error has been detected, and it will terminate with an error condition.

It is therefore wise to anticipate and avoid operations like division by zero that will cause errors. This is most easily done with the dollar control, and is discussed in section The Dollar Condition.
4.26.6 Predefined Symbols

GAMS offers several predefined symbols including: sameAs, diag, sortedUels, solvers, licenseCodes, solverCapabilities, componentEDate, componentMDate, and ComponentLicenseOptions. With the exception of sameAs, diag, and sortedUels these predefined symbols are for very special (and mostly internal) purposes, but the way they work is identical.

These symbols can be used in a GAMS program without being declared but work in an idiosyncratic way. For example, the complete program

display solvers, sameAs;

compiles and executes without error and produces the following result in the listing file:

---- 1 SET Solvers  Available Solvers
       ( EMPTY )

---- 1 SET SameAs  Set Element Comparison Without Checking
       ( EMPTY )

However, if labels have been declared (in any set) that corresponds to a solver name, then set solvers will contain this label. Consider the following example:

Set titles / Lord, Baron, Duke /;
Set greeks / Euclid, Pythagoras, Antigone /;
display solvers, sameAs;

Note that Antigone and Baron are the names of solvers in GAMS and therefore they will be displayed as a member of the set solvers. Also note that the set sameAs contains the diagonal elements for both sets:

---- 3 SET Solvers  Available Solvers
       Baron, Antigone

---- 3 SET SameAs  Set Element Comparison Without Checking
       Lord Baron Duke Euclid Pythagoras Antigone
       Lord YES
       Baron YES
       Duke YES
       Euclid YES
       Pythagoras YES
       Antigone YES

User defined symbols can have the same name as a predefined symbol. In this case the user symbol hides the predefined symbols as demonstrated in the following example:

Set diag / 1*3 /;
$if setType diag $log diag is a set
$if preType diag $log diag is a predefined type

The log will only contain the line diag is a set. The test $if preType diag fails.

The list of all predefined symbols can be retrieved by declaring and displaying a set using some system data

set pre /system.predefinedSymbols/; display pre; which results in

---- 1 SET pre
       SameAs , Diag , Solvers
       LicenseCodes , SolverCapabilities , SortedUels
       ComponentEDate , ComponentMDate , ComponentLicenseOptions
4.26.7 Summary

GAMS provides powerful facilities for data manipulation with parallel assignment statements, built-in functions and extended range arithmetic.

4.27 Data Entry: Parameters, Scalars and Tables

4.27.1 Introduction

Data handling is of crucial importance in all modeling applications. The quality of the numbers and the intelligence with which they are used is likely to be at least as important as the logic of the model in determining if an application is successful or not. GAMS has been designed to have a complete set of facilities for entering information, manipulating it and reporting on the results. In this chapter we will concentrate on data entry. Chapter Data Manipulations with Parameters introduces and discusses data manipulations. For details on reporting, see chapters GAMS Output, The Display Statement, The Put Writing Facility, and GAMS Data eXchange (GDX).

One very important principle will motivate all our discussions on data:

Note

Data should be entered in its most basic form and each data item should be entered only once.

There are two reasons for adopting this principle. Numbers are almost certain to change, and when they do we want to be able to make the process of changing them as easy and safe as possible. We also want to make our model easy for others to read and understand. Keeping the amount of data as small as possible will certainly help. All the data transformations are shown explicitly in the GAMS representation, which makes it possible to reproduce the results of a study and shows the reader all the assumptions made during data manipulation. Another advantage is that everything needed to run or change the model is included in one program that can easily be moved from place to place or from one machine to another.

This chapter deals with the data type parameter. For other data types, see section Data Types and Definitions. Data for parameters can be entered in three basic formats: scalar, list oriented, or tables of two or more dimensions. For each of these formats, GAMS offers a separate keyword:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>Single (scalar) data entry.</td>
</tr>
<tr>
<td>Parameter</td>
<td>List oriented data, defined over one or more sets.</td>
</tr>
<tr>
<td>Table</td>
<td>Table oriented data, must involve two or more dimensions.</td>
</tr>
</tbody>
</table>

Table 1: Parameters, Scalars and Tables

Note that the term parameter is used in two ways: as data type and as keyword, so one could also see scalars and tables as special formats of parameters. Each of the data input formats will be introduced and discussed in the following sections. At the end of the chapter the special data type acronym is introduced.

Note

• By default, parameters in all input formats may only be initialized once, thereafter data must be modified with assignment statements. This can be changed using the dollar control option $onMulti.
This chapter explains the complete syntax to declare parameters which includes the optional initialization. So, while it is possible to initialize the data at declaration, often the data is read from other sources like databases or spreadsheets. More information about this can be found in the chapter Data Exchange with Other Applications.

4.27.2 Scalars

The scalar statement is used to declare and (optionally) initialize a GAMS parameter of dimensionality zero. This means that there are no associated sets, so there is exactly one number associated with the parameter.

4.27.2.1 The Syntax

In general, the syntax for a scalar declaration in GAMS is as follows:

```
scalar[s] scalar_name [text] [/numerical_value/]
   { scalar_name [text] [/numerical_value/]} ;
```

The keyword `scalar[s]` indicates that this is a scalar statement and `scalar_name` is the internal name of the scalar in GAMS, it is an identifier. The optional explanatory text is used to describe the scalar and the optional numerical_value is assigned to be the value of `scalar_name`. Numerical_value can be given as fixed number or as constant evaluation. Alternatively, the special data type acronym may be used as value. For details on acronyms, see section Acronyms.

Note that more than one scalar may be declared in one scalar statement. The entries have to be separated by commas or by end of line. For advice on explanatory text and how to choose a `scalar_name`, see the tutorial Good Coding Practices.

Note that scalars may be declared but not initialized in the scalar statement. A value can also be assigned later as illustrated in the example that follows.

4.27.2.2 An Illustrative Example

An example of a scalar definition in GAMS is shown below.

```
Scalar
   rho "discount rate" / .15 /
   irr "internal rate of return"
   life "financial lifetime of productive units" / 20 /;
```

The statement above initializes rho and life, but not irr. Later on another scalar statement can be used to initialize irr or an assignment statement could be used to provide the value:

```
irr = 0.07;
```

For more on scalar assignments and parameter assignments in general, see section Data Entry by Assignment.
4.27.3 Parameters

The parameter format is used to enter list oriented data which can be indexed over one or several sets.

4.27.3.1 The Syntax

In general, the syntax for a parameter declaration in GAMS is as follows:

\[
\text{parameter}[s] \text{ param}_\text{name}[(\text{index\_list})] \text{ [text] } / \text{ element } = \text{ numerical\_value} \\
{,\text{element } = \text{ numerical\_value}} / \\
{,\text{param\_name}[(\text{index\_list})] \text{ [text] } / \text{ element } = \text{ numerical\_value} \\
{,\text{element } = \text{ numerical\_value}} /} ;
\]

The keyword parameter[s] indicates that this is a parameter statement and param_name is the internal name of the parameter in GAMS, it is an identifier. A parameter may be defined over one or more sets that may be specified in the index_list. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section Domain Checking. The optional explanatory text is used to describe the parameter.

Parameter initialization requires a list of data elements, each consisting of a label or label-tuple and a value. Element is an element of the defining set or - if there is more than one defining set - a combination of the elements of the defining sets. The referenced set elements must belong to the set that the parameter is indexed over. Finally, numerical_value is the value assigned to the record defined by the set element or element tuple. It can be given as fixed number or as constant evaluation. Alternatively, the special data type acronym may be entered as value. For details on acronyms, see section Acronyms.

Note

The default value of a parameter is 0.

Slashes must be used at the beginning and end of the list, and commas must be used if several data elements are listed in one line. An equals sign or a blank separates the label tuple from its associated value. A parameter can be defined in a similar syntax to that used for a set. For advice on explanatory text and how to choose a parameter name, see the tutorial Good Coding Practices.

Note

Several parameters may be declared in one parameter statement.

4.27.3.2 Illustrative Examples

The following example illustrates the parameter statement. It is adapted from [MEXSS]. We also show the set definitions because they make the example clearer. For more on sets definitions, see chapter Set Definition.

```
Set i "steel plants" / hylsa monterrey 
    hylsap puebla /

j "markets" / mexico-df, monterrey, guadalaja /;

Parameter
dd(j) "distribution of demand" 
    / mexico-df 55, 
    guadalaja 15 /;
```
The index specification for the parameter `dd` means that there will be a vector of data associated with it, one number corresponding to every member of the set `j`. The numbers are specified along with the declaration in a format very reminiscent of the way we specify sets: in this simple case a label followed by a blank separator and then a value. Any of the legal number entry formats are allowable for the value. For details on number formats in GAMS, see subsection Numbers. The default data value is zero. Since `monterrey` has been left out of the data list, the value associated with `dd('monterrey')` is zero. As with sets, commas are optional at end of line.

We may also list several data elements on a line, separated by commas as in the following example:

```
Parameter
   a(i) / seattle = 350, san-diego = 600 /
   b(i) / seattle 2000, san-diego 4500 /;
```

If a parameter is defined over a set and all elements of the set are assigned the same value, then the following notation may be used as a shortcut:

```
parameter param_name[(set_name)] [text] /(#|set.)set_name numerical_value/;
```

Here `set` is a reserved word and `set_name` is the name of the set as it has been declared in a previous set declaration statement. Instead of `set`, one could also use the `#` sign. The following artificial example illustrates this notation:

```
Set j /j1, j2/;
Parameter hh(j) /set.j 10/
    gg /#j 10/;
```

This resolves in `hh('j1') = hh('j2') = gg('j1') = gg('j2') = 10`.

Note

By default it is not possible to define an empty parameter at declaration. This may be changed using the dollar control option `$onEmpty`, as shown in the following example:

```
Set i / seattle, san-diego /;
$onEmpty
Parameter
   a(i) / /;
```

That initializes `a('seattle')` and `a('san-diego')` to 0. So it is not the same as this:

```
Set i / seattle, san-diego /;
Parameter
   a(i);
```

Here, `a` is declared, but not initialized (so, it is not defined yet) and one would get an error when trying to read it.
4.27.3.3 Parameter Data for Higher Dimensions

A parameter may have several dimensions. For the current maximum number of permitted dimensions, see Dimensions. The list oriented data initialization through the parameter statement can be easily extended to data of higher dimensionality. The label that appears on each line in the one-dimensional case is replaced by a label tuple for higher dimensions. The elements in the $n$-tuple are separated by dots (.) just like in the case of multi-dimensional sets.

The following example illustrates the use of parameter data for higher dimensions:

Parameter

\[
\text{salaries(employee,manager,department)}
\]

\[
/ \text{anderson .murphy .toy} = 6000 \\
\text{hendry .smith .toy} = 9000 \\
\text{hoffman .morgan .cosmetics} = 8000 /;
\]

All the mechanisms using asterisks and parenthesized lists that we introduced in our discussion of sets are available here as well. For details see section Multi-Dimensional Sets. Below is an artificial example, in which a very small fraction of the total data points are initialized. GAMS will mark an error if the same label combination (or label-tuple) appears more than once in a data list.

Set row / row1*row10 /
col / col1*col10 /;
Parameter

\[
a(row, col)
\]

\[
/ (row1,row4) . col2*col7 12 \\
\text{row10} . col10 17 \\
\text{row1*row7} . col10 33 /;
\]

In this example the twelve elements row1.col2 to row1.col7 and row4.col2 to row4.col7 are all initialized at 12, the single element row10.col10 at 17, and the seven elements rows1..col10 to row7.col10 at 33. The other 80 elements (out of a total of 100) remain at their default value, which is 0. This example shows the ability of GAMS to provide a concise initialization or definition for a sparse data structure.

4.27.4 Tables

Tabular data can be declared and initialized in GAMS using a table statement. For two and higher-dimensional parameters this provides an easier and more concise method of data entry than the list based approach, since - at least in smaller tables - each label appears only once.

4.27.4.1 The Syntax

In general, the syntax for a table declaration in GAMS is as follows:

```
table table_name[(index_list)] [text] [EOL
    element { element } EOL
    {element numerical_value { numerical_value} EOL}
} ;
```
The keyword **table** indicates that this is a **table** declaration and **table_name** is the internal name of the table in GAMS, it is an **identifier**. The name of the parameter can be followed by the **index_list**. In the **index_list** the sets are specified over which the table is defined. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section Domain Checking. The optional **explanatory text** is used to describe the table, followed by EOL which means ”end of line”, a line break. **Element** is an element of one of the driving sets. More details follow below. **Numerical_value** is the value of the entry associated with the corresponding element combination. It can be given as fixed number or as **constant evaluation**. Alternatively, the special data type **acronym** may be used as value. For details on acronyms, see section Acronyms. For advice on explanatory text and how to choose a **table_name**, see the tutorial Good Coding Practices.

**Attention**

By default, the table statement is the only statement in the GAMS language that is not free format. This may be changed using the dollar control option $onDelim.

The following rules apply:

- The relative positions of all entries in a table are significant. This is the only statement where end of line (EOL) has meaning. The character positions of the numeric table entries must overlap the character positions of the column headings.

- The column section has to fit on one line.

- The sequence of values forming a row must be on the same line.

- The element definition of a row can span more than one line.

- A specific column can appear only once in the entire table.

The rules for building simple tables are straightforward. The components of the header line are

```
keyword - identifier - index_list - text
```

Note that the **index_list** and the **text** are optional. Labels are used on the top and the left to map out a rectangular grid that contains the data values. The order of labels is unimportant, but if domain checking has been specified (i.e. the **index_list** has been given in the first line of the table declaration) each label must match one in the associated set. Labels must not be repeated, but can be left out if the corresponding numbers are all zero or not needed. At least one blank must separate all labels and data entries. Blank entries imply that the default value (zero) will be associated with that label combination.

**Note**

- Tables must have at least two dimensions. For the current maximum number of permitted dimensions, see Dimensions.

- The **table** statement can also be terminated with a ; in the first line, so without entering data. In this case the parameter **table_name** would be declared but not initialized.

- In contrast to the **set**, **scalar**, and **parameter** statements, only one identifier may be initialized in a **table** statement, though multiple ones could be declared.
4.27.4.2 An Illustrative Example

In the following example a simple table is presented. It is adapted from [KORPET], the relevant set definitions are also given.

Set i "plants"
   / inchon, ulsan, yosu /
m  "productive units"
   / atmos-dist "atmospheric distillation unit"
   steam-cr "steam cracker"
   aromatics "aromatics unit"
   hydrodeal "hydrodealkylator" /;

Table ka(m,i) "initial cap. of productive units (100 tons per yr)"
    inchon  ulsan  yosu
atmos-dist   3702   12910  9875
steam-cr     517    1207
aromatics    181    148
hydrodeal    180  
;

In this example the row labels are drawn from the set m and those on the column from the set i. Note that the data for each row is aligned under the corresponding column headings. Entries that are not specified are assigned the default value zero.

Note

If there is any uncertainty about which column a number belongs to, GAMS will protest with an error message and mark the ambiguous entry.

Attention

Special care has to be taken, if tabs are used. The GAMS command line option TabIn controls the tab spacing. Note that this spacing might be different form the spacing that the editor is showing, hence the visible alignment might be different from the alignment that GAMS is actually using.

4.27.4.3 Continued Tables

If a table has too many columns to fit nicely on a single line, then the columns that don't fit may be continued on additional lines. We use the same example to illustrate:

Table ka(m,i) "initial cap. of productive units (100 tons per yr)"
    inchon  ulsan
atmos-dist   3702   12910
steam-cr     517
aromatics    181
hydrodeal    180
+yosu
atmos-dist   9875
steam-cr     1207
aromatics    148
;

The crucial item is the plus '+' sign above the row labels and to the left of the column labels in the continued part of the table. The row labels have been duplicated, except that hydrodeal has been left out, since it does not have any associated data. Tables may be continued as many times as necessary.
4.27.4.4 Tables with more than Two Dimensions

Tables may have more than two dimensions. For the current maximum number of permitted dimensions, see Dimensions. As usual, dots are used to separate adjacent labels and may be used in the row or column position. The label on the left of the row corresponds to the first set in the index list, and that on the right of each column header to the last. Obviously, there must be the same number of labels associated with each number in the table, as there are sets in the index list.

The best layout depends on the size of the defining sets and the amount of data. It should provide the most intuitively satisfactory way of organizing and inspecting the data. For most people it is easier to look down a column of numbers than across a row. However, putting extra labels on the row has the advantage of greater density of information.

The following example, adapted from [MARCO], illustrates the use of tables with more than two dimensions.

Set ci "commodities : intermediate" / naphtha "naphtha" dist "distillate" gas-oil "gas-oil" / cr "commodities : crude oils" / mid-c "mid-continent" w-tex "west-texas" / q "attributes of intermediate products" / density, sulfur /;

Table attrib(ci, cr, q) "blending attributes"

<table>
<thead>
<tr>
<th></th>
<th>density</th>
<th>sulfur</th>
</tr>
</thead>
<tbody>
<tr>
<td>naphtha. mid-c</td>
<td>272</td>
<td>.283</td>
</tr>
<tr>
<td>naphtha. w-tex</td>
<td>272</td>
<td>1.48</td>
</tr>
<tr>
<td>dist . mid-c</td>
<td>292</td>
<td>.526</td>
</tr>
<tr>
<td>dist . w-tex</td>
<td>297</td>
<td>2.83</td>
</tr>
<tr>
<td>gas-oil. mid-c</td>
<td>295</td>
<td>.98</td>
</tr>
<tr>
<td>gas-oil. w-tex</td>
<td>303</td>
<td>5.05</td>
</tr>
</tbody>
</table>

The table attrib could also be laid out as shown below:

Table attrib (ci,cr,q) "blending attributes"

<table>
<thead>
<tr>
<th></th>
<th>w-tex.density</th>
<th>mid-c.density</th>
<th>w-tex.sulfur</th>
<th>mid-c.sulfur</th>
</tr>
</thead>
<tbody>
<tr>
<td>naphtha</td>
<td>272</td>
<td>272</td>
<td>1.48</td>
<td>.283</td>
</tr>
<tr>
<td>dist</td>
<td>297</td>
<td>292</td>
<td>2.83</td>
<td>.526</td>
</tr>
<tr>
<td>gas-oil</td>
<td>303</td>
<td>295</td>
<td>5.05</td>
<td>.98</td>
</tr>
</tbody>
</table>

4.27.4.5 Condensing Tables

All the mechanisms using asterisks and parenthesized lists that were introduced in the discussion of sets are available here as well. For details on these mechanisms, see section Multi-Dimensional Sets. The following example shows how repeated columns or rows can be condensed with asterisks and lists in parentheses. The set membership is not shown, but can easily be inferred.
Table upgrade(strat, size, tech)

<table>
<thead>
<tr>
<th>tech</th>
<th>small.tech1</th>
<th>small.tech2</th>
<th>medium.tech1</th>
<th>medium.tech2</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy-1</td>
<td>.05</td>
<td>.05</td>
<td>.05</td>
<td>.05</td>
</tr>
<tr>
<td>strategy-2</td>
<td>.2</td>
<td>.2</td>
<td>.2</td>
<td>.2</td>
</tr>
<tr>
<td>strategy-3</td>
<td>.2</td>
<td>.2</td>
<td>.2</td>
<td>.2</td>
</tr>
<tr>
<td>strategy-4</td>
<td>.2</td>
<td>.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table upgradex(strat, size, tech) "alternative way of writing table"

<table>
<thead>
<tr>
<th>tech1*tech2</th>
<th>strategy-1.(small,medium)</th>
<th>.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategy-2*strategy-3.(small,medium)</td>
<td>.2</td>
<td></td>
</tr>
<tr>
<td>strategy-4.medium</td>
<td>.2;</td>
<td></td>
</tr>
</tbody>
</table>

4.27.4.6 Handling Long Row Labels

It is possible to continue the row labels in a table on a second, or even third line in order to accommodate a reasonable number of columns. The break must come after a dot, and the rest of each line containing an incomplete row label-tuple must be blank.

The following example, adapted from [INDUS], is used to illustrate. This table actually has nine columns and many rows, here we have reproduced just a small part to show continued row label-tuples.

Table yield (c,t,s,w,z) "crop yield (metric tons per acre)"

<table>
<thead>
<tr>
<th>wheat.(bullock, semi-mech).la-plant. (heavy, january)</th>
<th>nwpf</th>
<th>pmw</th>
</tr>
</thead>
<tbody>
<tr>
<td>.385</td>
<td>.338</td>
<td></td>
</tr>
<tr>
<td>wheat.(bullock, semi-mech).la-plant. light</td>
<td>.506</td>
<td>.446</td>
</tr>
<tr>
<td>wheat.(bullock, semi-mech).la-plant. standard</td>
<td>.592</td>
<td>.524</td>
</tr>
<tr>
<td>wheat.(bullock, semi-mech).(qk-harv, standard). (heavy, january)</td>
<td>.439</td>
<td>.387</td>
</tr>
</tbody>
</table>

4.27.5 Constant Evaluation

Instead of fixed numerical values, one can also use constant expressions to assign values to parameters in a data statement. The syntax of constant expressions used in data statements follows the GAMS syntax as described in Data Manipulations with Parameters, but is restricted to scalar values and a subset of the GAMS intrinsic functions, as summarized below:

- Real numbers only
- Evaluation left to right
- Operator precedence:
  - ^ **
  - * /
  - + - binary and unary
  - < <= = <> >= > LE LE EQ NE GE GT
  - NOT
  - AND
  - OR XOR EQV IMP
• See Functions for list of supported functions

When used in a data statement, the constant expressions have to be enclosed in a pair of square brackets [ ] or curly brackets { }. Spaces can be used freely inside those brackets. Here is a little example:

```plaintext
Scalars x "PI half" / [pi/2] / 
e "famous number" / [ exp( 1 ) ] /;
```

```plaintext
Parameter y "demo" / USA.(high,low) [1/3]  
USA.medium {1/4} /;
```

### 4.27.6 Data Entry by Assignment

Data may also be entered using assignment statements. Assignments are introduced and discussed in detail in section The Assignment Statement. This section here is a short outlook and shows how parameters that have already been declared may be assigned values. The general assignment statement has the following form:

```plaintext
parameter_name[(index_list)] = expression;
```

Here `parameter_name` is the name of a parameter that has been declared previously in a scalar, parameter or table statement, `index_list` indicates the controlling indices and may either contain a set or sets, a label or label tuple or a combination of those, and `expression` may be a number, a numerical expression or an acronym. For details on numerical expressions, see section Expressions.

The following examples illustrate how assignments may be used for data entry.

```plaintext
Set j / j1, j2, j3 /;
Scalar a1;
Scalars a2 / 11 /;
Parameter cc(j), bc(j) / j2 22 /;

a1 = 10;
a2 = 5;
cc(j) = bc(j)+10;
cc("j1") = 1;
```

The scalar `a1` is declared but not initialized in the first scalar statement. It is assigned the value of 10 in the first assignment. The scalar `a2` is initialized in the second scalar statement and this value is changed to 5 in the second assignment. Note that the original data is not retained. In the parameter statement the parameter `cc(j)` is declared but not initialized and the parameter `bc(j)` is only initialized for `j2`. This means that `bc('j2') = 22` and `bc('j1') = bc('j3') = 0`, the default value. Now, the third assignment sets the parameter `cc(j)` and assigns to all elements of the set `j` the value of the parameter `bc(j)` plus 10. So we have `cc('j2') = 32` and `cc('j1') = cc('j3') = 10`. Note that in this example the set `j` has only three elements so only 3 assignments are made simultaneously. However, suppose that the number of set elements is large, say 100,000, then to each element a value is assigned with just one assignment statement. Finally, the value of `cc('j1')` is changed to 1.

Observe that in the examples above assignments either refer to one specific set element or to the whole set. It is also possible to make assignments to only a part of the set. The mechanisms for partial set references are discussed in section Restricting the Domain in Assignments. Set elements that are not assigned new values in an assignment with a partial set reference retain their previous values. Recall that these may be the default value, values from the parameter or table statement, or values resulting from previous calculations.
4.27 Data Entry: Parameters, Scalars and Tables

4.27.7 Acronyms

An acronym is a special data type that allows the use of strings as values. Note that acronyms have no numeric values and are treated as character strings only.

4.27.7.1 The Syntax

The declaration for an acronym is similar to a set or parameter declaration. The basic format is as follows:

```
Acronym[s] acronym_name [text] {, acronym_name [text]};
```

The keyword `acronym[s]` indicates that this is an acronym statement and `acronym_name` is the internal name of the acronym in GAMS, it is an identifier. The optional explanatory text is used to describe the acronym. For advice on explanatory text and how to choose an `acronym_name`, see the tutorial Good Coding Practices.

Note that more than one acronym may be declared in one acronym statement. The entries have to be separated by commas or by end of line. A simple example illustrates this:

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
```

4.27.7.2 Acronym Usage

Acronyms may be used as data in scalar, parameter and table statements. An example for acronyms in a parameter statement follows.

```
Set machines / m-1*m-5 /;
Acronym
    Monday, Tuesday, Wednesday, Thursday, Friday;
Parameter
    shutdown(machines)
    / m-1 Tuesday
    m-2 Wednesday
    m-3 Friday
    m-4 Monday
    m-5 Thursday /;
```

Acronyms may also be used in assignments as in the example below. For more on assignments, see section The Assignment Statement.

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek;
dayOfWeek = Wednesday;
```

Note that numerical operations like addition or subtraction are not allowed with acronyms. Such operations would be meaningless since acronyms do not have numeric values.

Another context where acronyms may be used is in logical conditions. For more on logical conditions, see chapter Conditional Expressions, Assignments and Equations. This is shown in the following example:
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek
  workHours /6/;
dayOfWeek = Wednesday;
workHours$(dayOfWeek <> Friday) = 8;

Note that only the equality and inequality operators may be used with acronyms. Other operations like addition and division are meaningless since acronyms do not have numeric values.

Acronyms are specific to GAMS and hence difficult to deal with when exchanging data with other systems. Users often replace parameters that contain acronyms with dynamic sets that have an additional index whose values correspond to the acronyms found in the original parameter. The machine shutdown data from above can be represented via a two-dimensional set as follows:

Set machines / m-1*m-5 /;
  weekdays / Monday, Tuesday, Wednesday, Thursday, Friday /
  shutdown(machines,weekdays) / m-1.Tuesday
   m-2.Wednesday
   m-3.Friday
   m-4.Monday
   m-5.Thursday /;

4.27.8 Summary

In this chapter, the declaration and initialization of parameters with the Scalar, Parameter, and Table statement have been discussed. Chapter Data Manipulations with Parameters will describe how this data can be changed with assignment statements.

4.28 Variables

4.28.1 Introduction

This chapter covers the declaration and manipulation of GAMS variables. Many of the concepts covered in the previous chapters are directly applicable here.

A variable is the GAMS name for what are called endogenous variables by economists, columns or activities by linear programming experts, and decision variables by industrial Operations Research practitioners. They are the entities whose values are generally unknown until after a model has been solved. A crucial difference between GAMS variables and columns in traditional mathematical programming terminology is that one GAMS variable is likely to be associated with many columns in the traditional formulation.

4.28.2 Variable Declarations

A GAMS variable, like all other identifiers, must be declared before it may be referenced.
4.28.2.1 The Syntax

The declaration of a variable is similar to a set or parameter declaration, in that domain lists and explanatory text are allowed and recommended, and several variables may be declared in one statement. The syntax is given below.

[var_type] variable[s] var_name [(index_list)] [text] [/var_data/] {, var_name [(index_list)] [text]}

The keyword var_type denotes the optional variable type that is explained in detail in the next subsection. Variable[s] is the keyword that indicates that this is a variable statement. Var name is the internal name of the variable in GAMS, it is an identifier. In the optional index list the set or sets may be specified over which an indexed variable is declared. The optional explanatory text may be used to describe the variable for future reference and to ease readability. Specifying variable data is another optional element in the variable statement. Variable data allows to initialize variable attributes at compile time. For an example and details on variable attributes, see section Variable Attributes.

A typical variable statement adapted from the model [RAMSEY] is shown below for illustration:

Variables
k(t) capital stock (trillion rupees)
c(t) consumption (trillion rupees per year)
i(t) investment (trillion rupees per year)
utility utility measure;

The declaration of k above implies, as usual, that references to k are restricted to the domain of the set t. A model that includes k will probably have several corresponding variables in the associated mathematical programming problem: most likely one for each member of t. In this way, very large models can be constructed using a small number of variables. (It is quite unusual for a model to have as many as 50 distinct variables.) It is still unclear from the declaration whether utility is not domain checked or whether it is a scalar variable, i.e., one without associated sets. Later references will be used to settle the issue. For more details on domain checking, see section Domain Checking.

As the syntax indicates, the explanatory text is optional. However, it is important that variable declarations include explanatory text and that this be as descriptive as possible, since the text is used to annotate the solution output. Note the use of 'per' instead of '/' in the text above: slashes are illegal in all unquoted text.

Note
- Variable names, the contained set element names plus the explanatory text must obey the general rules for language items.
- Variables can be defined over from 0 up to 20 sets
- The sets over which variables are declared indicate that these variables are potentially defined for every element of the defining sets. However the actual definition of variables does not occur until variables appear in an equation definition where the equation needs to be part of a model that in turn occurs in a solve statement.

4.28.2.2 Variable Types

There are nine basic types of variables that may be used in variable statements. These are shown in table Table 1.
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Default Bound</th>
<th>Lower Bound</th>
<th>Default Upper Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>free (default)</td>
<td>No bounds on variable. Both bounds may be changed from the default values by the user.</td>
<td>-inf</td>
<td></td>
<td>+inf</td>
<td></td>
</tr>
<tr>
<td>positive or nonnegative</td>
<td>No negative values are allowed for variable. The user may change both bounds from the default value.</td>
<td>0</td>
<td></td>
<td>+inf</td>
<td></td>
</tr>
<tr>
<td>negative</td>
<td>No positive values are allowed for variables. The user may change both bounds from the default value.</td>
<td>-inf</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>Discrete variable that can only take values of 0 or 1. For details see section Types of Discrete Variables. In relaxed Model types the integrality requirement is relaxed.</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>Discrete variable that can only take integer values between the bounds. The user may change both bounds from the default value. The default upper bound inside GAMS is +inf but when the variable is passed on to the solver, the option or command line parameterIntVarUpdecides what upper bound (by default +inf) is passed on to the solver in case GAMS has upper bound +inf. In relaxed Model types the integrality requirement is relaxed.</td>
<td>0</td>
<td></td>
<td>+inf</td>
<td></td>
</tr>
<tr>
<td>sos1</td>
<td>A set of variables, such that at most one variable within a group may have a non-zero value. For details see section Types of Discrete Variables.</td>
<td>0</td>
<td></td>
<td>+inf</td>
<td></td>
</tr>
<tr>
<td>Keyword</td>
<td>Description</td>
<td>Default Lower Bound</td>
<td>Default Upper Bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sos2</td>
<td>A set of variables, such that at most two variables within a group may have non-zero values and the two non-zero values are adjacent. For details see section Types of Discrete Variables.</td>
<td>0</td>
<td>+inf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>semicont</td>
<td>Semi-continuous, must be zero or above a given minimum level. For details see section Types of Discrete Variables.</td>
<td>1</td>
<td>+inf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| semiint | Semi-integer, must be zero or above a given minimum level and integer. For details see section Types of Discrete Variables.  
The default upper bound inside GAMS is +inf but when the variable is passed on to the solver, the option or command line parameter IntVarUp decides what upper bound (by default +inf) is passed on to the solver in case GAMS has upper bound +inf. In relaxed Model types the integrality requirement is relaxed. | 1 | +inf |

**Table 1:** Variable types and default bounds

The default type is **free**, which means that if the type of the variable is not specified, it will not be bounded at all. The most frequently used types are **free** and **positive**. The type **positive variables** is used for variables for which negative values are meaningless, such as capacities, quantities or prices. Note that bounds may be changed using variable attributes and assignment statements, see section **Variable Attributes**.

**Note**

- Every optimization model must contain at least one unrestricted named variable (i.e. one declared with the keywords **Variable** or **Free Variable**). This variable is the objective variable. Even an objective variable can have lower and upper bounds assigned via the .lo and .up variable attribute.

- If a model is unbounded, a frequent cause for the unboundedness is that the modeler forgot to make a variable positive.
4.28.2.3 Styles for Variable Declaration

Two styles are commonly used to declare variable types. The first is to list all variables with domain specifications and explanatory text as a group, and later to group them separately as to type. The example shown below is adapted from [MEXSS]. The default type is free, so phi and phipsi will be free variables in the example below. Note the use of variable names derived from the original mathematical representation.

**Variables**

- \( u(c,i) \) "purchase of domestic materials (mill units per yr)"
- \( v(c,j) \) "imports (mill tpy)"
- \( e(c,i) \) "exports (mill tpy)"
- \( \phi \) "total cost (mill us$)"
- \( \phi \text{ipsi} \) "raw material cost (mill us$)"

Positive Variables \( u, v, e \);

The commas in the list of positive variables are required separators.

**Attention**

It is possible to declare an identifier more than once. However, the second and any subsequent declarations should only add new information that does not contradict what has already been entered.

The second popular way of declaring variables is to list them in groups by type. We rewrite the example above using this second method:

**Free Variables**

- \( \phi \) "total cost (mill us$)"
- \( \phi \text{ipsi} \) "raw material cost (mill us$)"

Positive Variables

- \( u(c,i) \) "purchase of domestic materials (mill units per yr)"
- \( v(c,j) \) "imports (mill tpy)"
- \( e(c,i) \) "exports (mill tpy)"

The choice between the two approaches is best based on clarity.

4.28.3 Variable Attributes

While a GAMS parameter has one number associated with each unique label combination, a variable has several. They represent:

<table>
<thead>
<tr>
<th>Variable Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>.lo</td>
<td>Lower bound for the variable. Set by the user either explicitly or through default values associated with the variable type.</td>
</tr>
<tr>
<td>Upper bound</td>
<td>.up</td>
<td>Upper bound for the variable. Set by the user either explicitly or through default values associated with the variable type.</td>
</tr>
<tr>
<td>Fixed value</td>
<td>.fx</td>
<td>A fixed value for the variable. If set it results in the upper and lower bounds of the variable to be set to the value of the .fx attribute.</td>
</tr>
</tbody>
</table>
### Variable Attributes

<table>
<thead>
<tr>
<th>Variable Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity level</td>
<td>.l</td>
<td>Activity level for the variable, also the current value or starting point. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.</td>
</tr>
<tr>
<td>Marginal</td>
<td>.m</td>
<td>The marginal value (or reduced cost) for the variable. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.</td>
</tr>
<tr>
<td>Scale factor</td>
<td>.scale</td>
<td>Numerical scaling factor for all coefficients associated with the variable if the model attribute scaleopt is set to 1. For more on scaling, see section Model Scaling - The Scale Option. Only applicable for continuous variables.</td>
</tr>
<tr>
<td>Branching priority</td>
<td>.prior</td>
<td>Branching priority value used in mixed integer programming models if the model attribute prioropt is set to 1. For details see section Setting Priorities for Branching. It can also be used to relax discrete restrictions by setting .prior = +inf regardless of the prioropt setting. Only applicable for discrete variables.</td>
</tr>
<tr>
<td>Stage</td>
<td>.stage</td>
<td>This attribute allows to assign variables to stages in a stochastic program or other block structured model. Thus, among other places, it is used for 2-stage stochastic programs solved with DECIS or the Benders partition in Cplex.</td>
</tr>
</tbody>
</table>

#### Table 2: Variable Attributes

Users distinguish between these values when necessary by appending the suffix to the variable name. Examples are given below.

It is possible to specify initial values for these variable attributes at compile time. This can be done within the variable declaration statement as illustrated in the following example or during execution time as explained in section Assigning Values to Variable Attributes.

**Variable x1(j) my first / j1.up 10 , j1.lo 5, j1.l 7, j1.m 0, j1.scale 20 /;**

The upper bound of the variable x1("j1") is set to 10, the lower bound is set to 5, the starting value for the activity level is set to 7 and the starting value for the marginal is set to 0. The variable is also scaled by the factor 20, which means it is *multiplied* by 20.

Note that it is also possible to use a table structure to assign values to variable attributes. The following example is adapted from model [TRNSPORT].

**Variable Table x(i,j) initial values**

```plaintext
1  m
seattle. new-york 50
seattle. chicago 300
san-diego.new-york 275
san-diego.chicago 0.009;
```

---

**4.28 Variables** 617
Note

- `.fx` and attributes `.lo` and `.up` on the same variable cannot be in a data statement. `.fx` sets both `.lo` and `.up` and hence we would have a double definition of the same attribute. Since attribute `.scale` is applicable for continuous variables and attribute `.prior` for discrete variables, they share the same internal space in a GAMS variable. Some solvers can make use of priorities even for continuous variables (e.g. BARON). Such priorities need to be supplied via a solver option file.

- The attribute `.stage` uses the same internal space as `.scale` and `.prior`. So a model cannot specify scale factor and branching priorities together with stages.

- Fixing a semi-continuous or semi-integer variable to a non-zero value like `x.fx = 4` does not result in a truly fixed variable. The domain of the variable remains `{0,4}`. To really fix a semi-continuous or semi-integer variable, the discrete restriction could be relaxed by setting the branching priority to infinity (`x.prior=inf`).

- For variables in discrete models (such as MIP, MINLP), the `.m` attribute provides the marginals obtained by fixing all the discrete variables and solving the resulting continuous problem (such as LP, NLP). Many solvers allow to enable/disable solving such a fixed problem. When disabled, no marginals will be provided for discrete models.

In addition to the variable attributes introduced above, there are a number of variable attributes that cannot be assigned but may be used in computations. They are given in Table 3.

<table>
<thead>
<tr>
<th>Variable Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>.range</td>
<td>The difference between the lower and upper bounds for a variable. It becomes zero if the lower equals the upper bound, e.g. if the <code>.fx</code> attribute is set.</td>
</tr>
<tr>
<td>Slack upper bound</td>
<td>.slackup</td>
<td>Slack from variable upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of a variable.</td>
</tr>
<tr>
<td>Slack lower bound</td>
<td>.slacklo</td>
<td>Slack from variable lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of a variable.</td>
</tr>
<tr>
<td>Slack</td>
<td>.slack</td>
<td>Minimum slack from variable bound. This is defined as the minimum of two values: the slack from the variable lower bound and the slack from the variable upper bound.</td>
</tr>
<tr>
<td>Infeasibility</td>
<td>.infeas</td>
<td>Amount by which a variable is infeasible falling below its lower bound or above its upper bound. This is defined as the smallest of three values: zero, the difference between the lower bound and the level value, the difference between the level value and the upper bound of a variable, i.e. <code>max(0, lower-level, level-upper)</code>.</td>
</tr>
</tbody>
</table>

Table 3: Additional Variable Attributes that Cannot Be Assigned but May Be Used in Computations.

### 4.28.3.1 Bounds on Variables

All default bounds set at declaration time may be changed using assignment statements.

**Attention**

For discrete variable types, the consequences of the type declaration cannot be completely undone (e.g. the `.scale` attribute is not available) but their value domain can be changed to continuous by setting attribute `.prior` to infinity.
Bounds on variables are the responsibility of the user. After variables have been declared, default bounds have already been assigned: for many purposes, especially in linear models, the default bounds are sufficient. In nonlinear models, however, bounds play a far more important role. It may be necessary to provide bounds to prevent undefined operations, such as division by zero. In nonlinear programming it is often necessary to define a 'reasonable' solution space that will assist in efficiently finding a solution.

Attention

The lower bound cannot be greater than the upper bound: if you happen to impose such a condition, GAMS will generate an execution error, namely "**** Matrix error - lower bound > upper bound" when executing a solve statement.

Note that the upper bound on integer and semi-integer variables needs special consideration. The default upper bound is +inf and the option or command line parameter IntVarUp controls what upper bound is sent to the solver. With the current default value (0) of IntVarUp, an upper bound of +inf is sent to the solver. Setting IntVarUp to one will pass 100 as the default upper bound to the solver. The other available values for IntVarUp work like zero, but enable special reports/execution errors in case the solution reports a level value greater than 100 for any integer variable with a default bound of +inf.

4.28.3.2 Fixing Variables

GAMS allows the user to fix variables through the .fx variable suffix. This is almost equivalent to setting the lower bound and upper bound equal to the fixed value. The attribute .fx also resets the activity level .l to the fixed value. When setting .lo and .up the activity level remains unchanged. A solve statement will project the activity level within the active bounds. Fixed variables can subsequently be freed by changing the lower and upper bounds.

4.28.3.3 Activity Levels of Variables

GAMS allows the user to set the activity levels of variables through the .l variable suffix. These activity levels of the variables prior to the solve statement serve as initial value for the solver. This is particularly important for nonlinear programming problems. For discrete models in many cases the solver needs an additional indicator to interpret the activity levels as a feasible integer solution via a solver option (e.g. Cplex' mipstart).

Attention

- GAMS only stores variables with non-default values (similar to storing only non-zero values of parameters). Non-default variables can be accidentally created by using harmlessly looking assignments like
  
x.up(i,j,k,l) = 0;

  Even if the equations only reference such variables over a small subset of \((i,j,k,l)\) this statement creates \(\text{card}(i) \times \text{card}(j) \times \text{card}(k) \times \text{card}(l)\) variable records in the GAMS database. Such fixings of \(x(i,j,k,l)\) to 0 can be avoided by using dynamic sets or dollar conditions in the equation algebra to only reference tuples of \((i,j,k,l)\) for which \(x(i,j,k,l)\) can possible have a non-zero value.

- In order to filter only necessary tuples for an equation the filtering conditions needs to be provided only once when defining the equation (equ(1,j,k)). This is different for variables because they appear in many equations and the filtering condition needs to be potentially repeated many time. Therefore it is good practice and reduces GAMS model generation time if the filtering of the variables is governed by a dynamic set:

  \[
  \text{sum}((i,j)\{\text{ord}(i) \text{ord}(j) \text{cap}(i,j)>0\}, x(i,j))
  \]

  versus

  \[
  \text{set net(i,j); net(i,j) = ord(i)ord(j) cap(i,j)>0;}
  \text{sum(net(i,j), x(i,j))}
  \]
4.28.4 Variables in Display and Assignment Statements

GAMS allows the modeler to use the values associated with the various attributes of each variable in assignment and display statements. The next two subsections explain the use of variables on the left and right-hand sides of assignment statements respectively. Then we will explain the use of variables in display statements.

4.28.4.1 Assigning Values to Variable Attributes

Assignment statements operate on one variable attribute at a time, and require the suffix to specify which attribute is being used. Any index list comes after the suffix.

The following code snippets are from models [MEXSS] and [RAMSEY]. The first example illustrates the use of assignment statements to set upper and lower bounds on variables.

\[
x\.up(c, i, j) = 1000; \quad \text{phi\.lo} = \text{inf};
\]

A very common use is to bound one particular entry individually:

\[
p\.up('pellets', 'ahmsa', 'mexico-df') = 200;
\]

Or to put small lower bounds on a variable identifier used as a divisor in a nonlinear program:

\[
c\.lo(t) = 0.01;
\]

Or to provide initial values for a nonlinear problem:

\[
c\.l(t) = 4*c\.init(t);
\]

Remember that the order is important in assignments, and notice that the two pairs of statements below produce very different results. In the first case, the lower bound for \(c('1985')\) will be 0.01, but in the second, the lower bound is 1.

\[
c\.fx('1985') = 1; \quad c\.lo(t) = 0.01;
c\.lo(t) = 0.01; \quad c\.fx('1985') = 1;
\]

Everything works as described in the previous chapter, including the various mechanisms described there of indexed operations, dollar operations, subset assignments and so on. An example from model LOCATION follows.

\[
\text{ship\.sm\.lo}(sl,m)$\text{(ord(sl) = 1 and ord(m) = 1)} = 1;
\]

The lower bound of the variable \text{ship\.sm}(sl,m) is set to 1 and this assignment is only valid for \text{ship\.sm}('s1','d1'), the realization of the variable where both indices are the first members of their respective sets.
4.28.4.2 Variable Attributes in Assignments

Using variable attributes on the right-hand side of assignment statements is important for a variety of reasons. Two common uses are for generating reports and for generating initial values for some variables based on the values of other variables. For more on variable attributes in report writing, see section Displaying Variable Attributes below and especially chapter The Put Writing Facility.

The following examples adapted from model [CHENERY] illustrate the use of variable attributes on the right-hand side of assignment statements:

* initial values for variables

\[
\begin{align*}
y.l(i) &= 250 ; \quad x.l(i) = 200 ; \\
e.l(t) &= 0 ; \quad m.l(t) = 0 ; \\
g.l(t) &= \text{mew}(t) + \text{xsi}(t) \cdot m.l(t) ; \\
h.l(t) &= \text{gam}(t) - \text{alp}(t) \cdot e.l(t) ; \\
&\ldots
\end{align*}
\]

* generating report after solve

Scalar

\[
\begin{align*}
\text{cva} & \quad "\text{total value added at current prices}" \\
\text{rva} & \quad "\text{real value added}" \\
\text{cli} & \quad "\text{cost of living index}"
\end{align*}
\]

\[
\begin{align*}
\text{cva} &= \sum(i, v.l(i) \cdot x.l(i)) ; \\
\text{cli} &= \sum(i, p.l(i) \cdot \text{ynot}(i)) / \sum(i, \text{ynot}(i)) ; \\
\text{rva} &= \text{cva}/\text{cli} ;
\end{align*}
\]

Display cli, cva, rva ;

As with parameters, a variable must have some non-default data values associated with it before it can be used in a display statement or on the right-hand side of an assignment statement. After a solve statement has been processed or if non-default values have been set with an assignment statement, this condition is satisfied. Solve statements are introduced and discussed in chapter Model and Solve Statements.

Attention

The .fx suffix is mostly just a shorthand for .lo and .up and can therefore only be used only on the left-hand side of an assignment statement.

Note

- In general, the variable level needs to be specified via the attribute .l for assignment statements. However, the dollar control option $on/offDotL allows the implicit use of the attribute .l in assignment statements, thus it facilitates using the same algebra in model definitions and assignment statements. This is especially useful in the context of macros.
4.28.4.3 Displaying Variable Attributes

The display statement is introduced and discussed in detail in chapter The Display Statement. Here we demonstrate how variable attributes are used in display statements.

Since several values are associated with each variable, the user must specify which attribute should be displayed when using variables in display statements. As before, appending the appropriate suffix to the variable name does this and no domain specification may appear. As an example, we show how to display the level of $\phi$ and the level and the marginal values of $v$ from [MEXSS]:

```
display phi.l, v.l, v.m;
```

The output looks similar, except that (of course) the listing shows which of the values is being displayed. Because zeroes, and especially all zero rows or columns, are suppressed, the patterns seen in the level and marginal displays will be quite different, since non-zero marginal values are often associated with activity levels of zero.

Mexico Steel - Small Static (MEXSS,SEQ=15)

```
Execution

---- 203 VARIABLE PHI.L = 538.811 total cost 
      (mill us$)
---- 203 VARIABLE V.L imports 
      (mill tpy) 
      ( ALL 0.000 )
---- 203 VARIABLE V.M imports 
      (mill tpy) 
      mexico-df monterrey guadalaja 
      steel 7.018 18.822 6.606

We should mention here a clarification of our previous discussion of displays. It is actually the default values that are suppressed on display output. For parameters and variable levels and marginals, the default is zero, and so zero entries are not shown. For bounds, however, the defaults can be non-zero. The default value for the upper bound of a positive variable is $+\text{INF}$, and if we would also display $v.\text{up}$ above, for example, we would see:

```
---- 203 VARIABLE V.UP imports 
      (mill tpy) 
      ( ALL $+\text{INF}$ )
```

If any of the bounds have been changed from the default value, then only the entries for the changed elements will be shown. This may sound confusing, but since few users display bounds it has not proved troublesome in practice.

Note

The attribute .range may be used in display statements. It provides a convenient way to check whether a variable is fixed as it lists the 0 values explicate because the default for range (which won’t be displayed) is $+\text{inf}$.
4.28.5 Summary

Remember that wherever a parameter may appear in a display or an assignment statement, a variable may also appear - provided that it is qualified with one of the suffixes. The only places where a variable name may appear without a suffix is in a variable declaration, as has been shown in this chapter, in an equation definition, which is discussed in chapter Equations, or in a $on/offDotL block.

4.29 Equations

4.29.1 Introduction

The keyword equation defines GAMS names that may be used in the model statement. A GAMS equation name is associated with the symbolic algebraic relationships that will be used to generate the constraints in a model. The algebraic relationships are defined by using constants, mathematical operators, functions, sets, parameters and variables. As with variables, one GAMS equation may be defined over a group of sets and in turn map into several individual constraints associated with the elements of those sets. Most of the example code in this chapter is from the model location.

This chapter is organized as follows. First, we introduce how equations are declared and defined, then we discuss expressions in equation definitions, followed by a section on equation attributes. A summary and quick reference conclude the chapter.

4.29.2 Declaring Equations

An equation must be declared before it can be defined and used in a model.

4.29.2.1 The Syntax

The declaration of an equation is similar to a set or parameter declaration. The syntax is given below.

Equation[s] eqn_name [(index_list)] [explanatory text] [/eqn_data/] {, eqn_name [(index_list)] [explanatory text] [/eqn_data/]} ;

Equation[s] is the reserved word that indicates that one or more blocks of equations are about to be declared. A block of equations may initiate one or more individual constraints. Eqn_name is the internal name of the equation, an identifier in GAMS. In the optional index_list the set or sets are specified over which an indexed equation is declared. The optional explanatory text may be used to describe the equation for future reference and to ease readability. Specifying equation data is another optional element in the equation declaration. Equation data allows to initialize equation attributes at compile time. For an example see the next section. For more on equation attributes see section Equation Attributes.

One or more equations may be declared in one equation statement. The equation names have to be separated by commas or by a line break as in the example that follows. The end of the declaration statement is indicated by a semicolon.

Note

It is good practice to end the equation declaration with a semicolon, even though it is not mandatory if the next statement starts with a GAMS keyword.

For advice on choosing equation names and phrasing the explanatory text see chapter Good Modeling Practices.
4.29.2.2 An Illustrative Example

The following example is from the model location. In addition to the equation declarations the relevant set definitions are given.

Sets
    sl 'supply locations' /s1, s2/
    wh 'warehouse locations' /a, b, c/;

Equations
    tcost_eq 'total cost accounting equation'
    supply_eq(sl) 'limit on supply available at supply location'
    capacity_eq(wh) 'warehouse capacity' /a.scale 50, a.l 10, b.m 20/;

The keyword Equations marks the beginning of the equation declaration. Each equation name is optionally followed by its domain (associated set or sets) unless it is a scalar equation. It is possible but not good practice to declare indexed equations without their domains. The name of the first equation is tcost_eq and it is followed by the explanatory text 'total cost accounting equation'. The name of the equation tcost_eq is not followed by any associated sets. Since we follow good practice here we assume that tcost_eq is a scalar equation. Scalar equations do not have any associated sets and will generally produce one equation in the model. For more on scalar equations see subsection Scalar Equations.

The other two equations are indexed equations, they are declared over a set. The equation supply_eq is declared over the set sl and the equation capacity_eq is declared over the set wh. In typical circumstances an indexed equation declaration implies that a block of constraints will be generated. For example, equation supply_eq(sl) implies that two constraints will be generated, one for each element of the set sl. For more on indexed equations see subsection Indexed Equations.

The declaration of the equation capacity_eq specifies some equation attributes. The first entry indicates that the equation capacity_eq('a') is scaled by a factor of 50, which means division of all entries in that equation by 50 upon model passage to the solver. For more on scaling see section Model Scaling - The Scale Option. The second entry sets the initial value of the equation capacity_eq('a') to 10 and b.m means that the initial marginal value of the equation capacity_eq('b') is set to 20. Alternatively, a table structure may be used to specify the values of equation attributes. The following table may replace the notation above.

Equation Table capacity_eq(wh) 'warehouse capacity'
        scale  l  m
    a      50   10  
    b         20 ;

For more on equation attributes see section Equation Attributes.

Note

An equation may be declared over more than one set.

4.29.3 Defining Equations

After declaring equations they have to be defined. The definition of an equation specifies the algebraic structure of the equation in GAMS. The syntax is given first, an illustrative example follows and in the remainder of this section some of the key components of equation definitions are discussed.
4.29.3.1 The Syntax

The syntax for defining an equation is as follows:

```
eqn_name(index_list)[$logical_condition(s)].. expression eqn_type expression ;
```

Eqn_name is the name of the equation as introduced in the equation declaration, that may be followed by an index_list for indexed equations. In the index_list the set or sets are specified over which an indexed equation is defined. These sets are also called domain of definition of the equation. One or more logical conditions are optional. For an example see Indexed Equations. For more on logical conditions in equation definitions see Dollar Control over the Domain of Definition. The two dots '..' are mandatory and indicate the start of the algebra. It is good practice to end the definition of an equation with a semicolon, even though it is not mandatory if the next statement starts with a GAMS keyword.

Attention

An equation must be declared before it is defined.

Expression refers to an algebraic expression which may include variables, parameters, functions, and constants among other items. For details on expressions in GAMS, see section Expressions.

Attention

Only variables that appear at least once with a nonzero coefficient in an equation definition will appear in a model.

Eqn_type refers to the equation type denoted by the symbol between the right-hand side and left-hand side expressions that form the equation. The symbols that are allowed are given in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=e=</td>
<td>Equality: right-hand side must equal left-hand side.</td>
</tr>
<tr>
<td>=g=</td>
<td>Greater than: left-hand side must be greater than or equal to right-hand side.</td>
</tr>
<tr>
<td>=l=</td>
<td>Less than: left-hand side must be less than or equal to right-hand side.</td>
</tr>
<tr>
<td>=n=</td>
<td>No relationship implied between left-hand side and right-hand side. This equation type is ideally suited for use in MCP models and in variational inequalities.</td>
</tr>
<tr>
<td>=x=</td>
<td>Equation is defined by external programs. See External Equations.</td>
</tr>
<tr>
<td>=c=</td>
<td>Conic constraint. See Conic Programming in GAMS.</td>
</tr>
<tr>
<td>=b=</td>
<td>Boolean equations. See Logic Equations.</td>
</tr>
</tbody>
</table>

Table 1: Equation Types

Equation definitions may be carried over as many lines of input as needed. Blanks may be inserted to improve readability, and expressions may be arbitrarily complicated.

Note that an equation can only be defined once. By using logical conditions it is possible to control which constraints are generated. In addition, the components of an equation may be modified by changing the data it uses. However, if the logic of the equation needs to be changed then a new equation with a new name has to be declared and defined.
4.29.3.2 An Illustrative Example

Consider the following example adapted from the model [mexss]. The associated variable and equation declarations are also included.

Variables phi, phipsi, philam, phipi, phieps;
Equations obj;
obj.. phi =e= phipsi + philam + phipi - phieps;

The name of the equation being defined is obj and the symbol =e= indicates that this is an equality. Any of the following forms of the equation are mathematically equivalent.

\[
\begin{align*}
\text{obj.. } & \phi + \phi_i + \phi_p - \phi = \phi_i \\
\text{obj.. } & \phi_i - \phi = \phi_p + \phi_i - \phi \\
\text{obj.. } & 0 = \phi + \phi_i - \phi_p - \phi_i + \phi_i - \phi_p \\
\end{align*}
\]

Note

The arrangement of the terms in the equation is a matter of choice, but often a particular one is chosen because it makes the model easier to understand.

4.29.3.3 Scalar Equations

A scalar equation will produce one equation in the associated optimization problem. The equation obj defined above is an example of a scalar equation which contains only scalar variables. Note that scalar equations may contain indexed variables. However, they must occur with an indexed operator such as sum or prod, unless the indexed variables refer to a singleton set (a set with only one element). Consider the following example from the model location. Note that the set wh has three elements.

\[
\text{configure_eq.. } \sum(\text{wh}, \text{build(wh)}) =l= 1;
\]

The variable build is defined over the set wh, it is an indexed variable. It may be used in the scalar equation configure_eq since it occurs in conjunction with the indexed operator sum.

4.29.3.4 Indexed Equations

All the set references in scalar equations are within the scope of indexed operators or they refer to singleton sets; thus many variable, set and parameter references can be included in one equation. In addition, GAMS also allows for equations to be defined over a domain, thereby developing a compact representation for constraints. The index sets to the left of ‘..’ are called the domain of definition of the equation.
4.29 Equations

Note

• Domain checking ensures that the domain over which an equation is defined is the set (or the sets) or a subset of the set (or the sets) over which the equation was declared.

• As a corollary, domain checking also catches the error of the indices being listed in an inconsistent order. For example, declaring an equation as `myequation(s,t)` and then naming it in the definition as `myequation(t,s)` causes an error (unless `s` and `t` are aliases of the same set). For more information, see section Domain Checking.

The following indexed equation with a single index generates a separate constraint for each member of the driving (or controlling) set. It is taken from the model [chenery]. In this example, `t` is a set with three members, `mew` and `xsi` are parameters and `m` and `g` are variables.

```plaintext
dg(t).. g(t) =e= mew(t) + xsi(t)*m(t) ;
```

As the set `t` has three members, three constraints will be generated, one for each member of `t` specifying the dependence of `g` on `m`. The data associated with the parameters `mew` and `xsi` are used to build the individual constraints. This data does not have to be known when the equation is defined, but it has to be populated before a model containing the equation is solved.

The extension to two or more indices on the left of `..` is obvious. There will be one constraint generated for each combination of set elements that can be constructed using the indices inside the parenthesis. Here are two examples from the model [aircraft], a scheduling model.

```plaintext
bd(j,h).. b(j,h) =e= dd(j,h) - y(j,h) ;
yd(j,h).. y(j,h) =l= sum(i, p(i,j)*x(i,j)) ;
```

The domain of definition of both equations is the Cartesian product of `j` and `h`: constraints will be generated for every set element pair that can be constructed from the members of these two sets.

The next example illustrates the use of the optional logical conditions in the definition of equations. It is taken from the production and distribution model [ferts].

```plaintext
CC(m,i)$mpos(m,i).. sum(p$ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i) ;
```

`CC` is a capacity constraint defined for elements of the sets `m` and `i`. However, in this case not all cases of `m` exist at each location `i`, and the mapping set `mpos(m,i)` tells the cases where `m` exists at `i` and thus is used to restrict the domain cases for which the constraints are actually generated. The control of the summation over `p` with `ppos(p,i)` is an additional logical condition, and is required because not all processes `p` are possible at all locations `i`.

The equation may alternatively written in the following way:

```plaintext
CC(mpos(m,i)).. sum(ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i) ;
```

Instead of defining the equation over the indices `(m,i)` the equation is defined over the set `mpos` that is itself defined over the indices `(m,i)`. A similar logic applies to restricting the summation.

Conditional expressions are introduced and discussed in the section Conditional Expressions, Assignments and Equations. See specifically Dollar Control over the Domain of Definition for logical conditions in equation definitions.
4.29.3.5 Using Labels Explicitly in Equations

Sometimes it can be necessary to refer to specific set elements in equations. This can be done as with parameters - by using quotes or double quotes around the label. Consider the following example from the model location:

\[ \text{sum}(m, \text{ship}_\text{wm}(\text{wh},m)) = \text{build}(\text{wh}) \times \text{data}(\text{wh},"capacity") \; ; \]

4.29.3.6 Logic Equations

Logic equations use Boolean algebra and have to evaluate to TRUE (or 1) to be feasible. The Boolean functions available in GAMS and the default order of precedence of the operators are given in Table 2. Note that 1 denotes the highest order of precedence or the most binding operator and 3 denotes the lowest order of precedence or the least binding operators. As usual, the default order of precedence holds only in the absence of parentheses and operators on the same level are evaluated from left to right.

<table>
<thead>
<tr>
<th>Function</th>
<th>Operator</th>
<th>Alternative Notation</th>
<th>Return Values</th>
<th>Order of precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>not x</td>
<td>bool_not(x)</td>
<td>returns 1 if ( x = 0 ), else returns 0</td>
<td>1</td>
</tr>
<tr>
<td>Logical conjunct</td>
<td>x and y</td>
<td>bool_and(x,y)</td>
<td>returns 1 if ( x = y = 1 ), else returns 0</td>
<td>2</td>
</tr>
<tr>
<td>Logical disjunct</td>
<td>x or y</td>
<td>bool_or(x,y)</td>
<td>returns 0 if ( x = y = 0 ), else returns 1</td>
<td>3</td>
</tr>
<tr>
<td>Exclusive disjunct</td>
<td>x xor y</td>
<td>bool_xor(x,y)</td>
<td>returns 1 if exactly one argument is 1, else returns 0</td>
<td>3</td>
</tr>
<tr>
<td>Material implica</td>
<td>x imp y</td>
<td>bool_imp(x,y)</td>
<td>returns 0 if ( x = 1 ) and ( y = 0 ), else returns 1</td>
<td>3</td>
</tr>
<tr>
<td>Material equiv</td>
<td>x eqv y</td>
<td>bool_eqv(x,y)</td>
<td>returns 0 if exactly one argument is 0, else returns 1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Boolean Functions and Operator Precedence

Note that in the table above \( x \) and \( y \) are binary variables.

There are three ways to declare and define logic equations:

1. The logic equation is declared using the keyword Logic Equation and the definition contains only Boolean algebra symbols.
2. The logic equation is declared like any other equation using the keyword Equation and in the definition the symbol \( =b= \) appears indicating that it is a logic equation.
3. This is a combination of the first two options: the equation is declared with the keyword Logic Equation and defined using the symbol \( =b= \).

The following example demonstrates the first way to declare and define a logic equation. It is adapted from the food manufacturing problem [foodemp]. In this problem the blending of oils is modeled.
Sets
  m        "planning period (month)" / m1*m6 /
p        "raw oils"         / v1*v2, o1*o3 /

Variables
  induse(m,p) "indicator for usage of raw oil per month" ;
  Binary variable induse;

Logic Equation
  deflogic(m) "if some vegetable raw oil is used we also need to use the non-vegetable oil o3" ;
  deflogic(m).. induse(m,'v1') or induse(m,'v2') -> induse(m,'o3');

The variable induse is a binary variable, it can only take the values 0 and 1. The equation ensures that in an optimal solution if either vegetable oil v1 or vegetable oil v2 is blended in a product, then non-vegetable oil o3 is also blended in that product.

An alternative formulation of the equation deflogic using the =b= notation is given below.

  deflogic(m).. induse(m,'v1') or induse(m,'v2') -> induse(m,'o3') =b= 1;

Note that the value of 1 on the right-hand side means that the logic expression on the left-hand side must evaluate to TRUE in a feasible solution. To illustrate further, we could negate the left-hand side expression using the logic operator not and then the right-hand side would have to evaluate to zero or FALSE to yield the same result as above. The respective equation definition follows.

  deflogic(m).. not (induse(m,'v1') or induse(m,'v2') -> induse(m,'o3')) =b= 0;

Note that currently logic equations are allowed only in models of the GAMS model type EMP. For more on GAMS model types see section Classification of Models.

4.29.4 Expressions in Equation Definitions

The arithmetic operators and some of the functions that are described in section Expressions may be used in equation definitions.

Consider the following example adapted from the model [chenery] demonstrating the use of parentheses and exponentiation.

  dem(i) ..   y(i) =e= ynot(i)*(pd*p(i))**thet(i) ;

A list of arithmetic operators is given in subsections Standard Arithmetic Operations and Indexed Operations.
4.29.4.1 Functions in Equation Definitions

All available GAMS functions are listed in the section Functions. Some functions are not allowed at all in equation definitions. They include random distribution functions and are marked with none in the third column of the tables listing all functions.

Attention

Some functions like uniform and normal are not allowed in equation definitions.

The use of the other functions is determined by the type of arguments in the model. There are two types of arguments:

1. **Exogenous arguments:** The arguments are known. Parameters and variable attributes (for example, .l and .m attributes) are used as arguments. The expression is evaluated once when the model is being set up and most mathematical functions as well as time and calendar functions are allowed.

2. **Endogenous arguments:** The arguments are variables and therefore unknown at the time of model setup. The function will be evaluated many times at intermediate points while the model is being solved. Note that the occurrence of any function with endogenous arguments implies that the model is not linear.

Functions that are allowed only with exogenous arguments are marked with any in the tables listing all functions.

There are two types of functions allowing endogenous arguments: smooth functions and discontinuous functions. Smooth functions are continuous functions with continuous derivatives (like sin, exp, log). Discontinuous functions include continuous functions with discontinuous derivatives (like max, min, abs) and discontinuous functions (like ceil, sign). Smooth functions may be used routinely in nonlinear models. However, discontinuous functions may cause numerical problems and should be used only if unavoidable, and only in a special model type called DNLP. For more details on model types see section Classification of Models.

Attention

The best way to model discontinuous functions is with binary variables. The result is a model of the type MINLP. The model [ABSMIP] demonstrates this formulation technique for the functions abs, min, max and sign. See also section Reformulating DNLP Models. We strongly discourage the use of the DNLP model type.

In Table 3 the use of functions in equation definitions is summarized.

<table>
<thead>
<tr>
<th>Functions are allowed ...</th>
<th>Description of Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>not at all</td>
<td>Functions that are marked none in the third column of the tables listing all functions in section Functions.</td>
</tr>
<tr>
<td>only with exogenous arguments</td>
<td>Functions that are marked any in the third column of the tables listing all functions in section Functions.</td>
</tr>
<tr>
<td>with endogenous arguments</td>
<td>Smooth functions. They are marked NLP in the third column of the tables listing all functions in section Functions.</td>
</tr>
<tr>
<td>with endogenous arguments</td>
<td>Discontinuous functions. They are marked DNLP in the third column of the tables listing all functions in section Functions.</td>
</tr>
</tbody>
</table>
4.29 Equations

Table 3: Functions in Equation Definitions

4.29.4.2 Preventing Undefined Operations in Equations

Some operations are not defined at particular values of the arguments. Two examples are division by 0 and the \( \log \)- of 0. While this can easily be identified at model setup for exogenous functions and expressions, it is a lot more difficult when the terms involve variables. The expression may be evaluated many times when the problem is being solved and the undefined result may arise only under certain cases. One way to avoid an expression becoming undefined is adding bounds to the respective variables. Consider the following example from the model [ramsey]:

\[
c_{lo}(t) = 0.01 ;
\]

\[
\text{util . utility} = \text{e= sum(t, beta(t)*log(c(t))) ;}
\]

Specifying a lower bound for \( c(t) \) that is slightly larger than 0 prevents the \( \log \)-function from becoming undefined.

4.29.5 Equation Attributes

Equation attributes may be specified in a similar way as variable attributes. Five values are associated with each unique label combination of every equation. They are denoted by the suffixes \( .l \), \( .m \), \( .lo \), \( .up \) and \( .scale \). A list of the attributes and their description is given in Table 4.

<table>
<thead>
<tr>
<th>Equation Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>.lo</td>
<td>Negative infinity for ( =l= ) equations. Right hand side value for ( =g=, =e=, ) and ( =b= ) equations. Zero for ( =c= ) equations.</td>
</tr>
<tr>
<td>Upper bound</td>
<td>.up</td>
<td>Positive infinity for ( =g= ) and ( =c= ) equations. Right hand side value for ( =l=, =e=, ) and ( =b= ) equations.</td>
</tr>
<tr>
<td>Equation level</td>
<td>.l</td>
<td>Level of the equation in the current solution, equal to the level of all terms involving variables.</td>
</tr>
<tr>
<td>Marginal</td>
<td>.m</td>
<td>Marginal value for equation. This attribute is reset to a new value when a model containing the equation is solved. The marginal value for an equation is also known as the shadow price for the equation and in general not defined before solution but if present it can help to provide a basis for the model</td>
</tr>
<tr>
<td>Scale factor</td>
<td>.scale</td>
<td>Numerical scaling factor that scales all coefficients in the equation. This is only used when the model attribute ( scaleopt ) is set to 1. For more on scaling, see section ( Model Scaling - The Scale Option. )</td>
</tr>
<tr>
<td>Stage</td>
<td>.stage</td>
<td>This attribute allows to assign equations to stages in a stochastic program or other block structured model. Its current use is limited to 2-stage stochastic programs solved with DECIS.</td>
</tr>
</tbody>
</table>

Table 4: Equation Attributes

Note that all attributes except for \( .scale \) and \( .stage \) contain the attribute values of equations after a solution of the model has been obtained. For some solvers it can be useful to specify marginal values \( .m \) and level values \( .l \) on input to provide starting information. Also note that the marginal value is also known as the \( dual \) or \( shadow price. \) Roughly speaking, the marginal value \( .m \) of an equation is the amount by which the value of the objective variable would change if the equation level were moved one unit.
Equation attributes may be referenced in expressions and can be used to specify starting values (see section Declaring Equations). In addition, they serve for scaling purposes and for reporting after a model was solved. For example, they may be displayed using the display statement. The following example is from the model location.

Model warehouse 'warehouse location model' /all/;
solve warehouse using mip min tcost;
display supply_eq.l;

The display statement generates the following output at the end of the listing file:

---- 108 EQUATION supply_eq.l limit on supply available at supply location

s1 50.000, s2 75.000

The level values of the equation supply_eq are displayed. As expected, there are two level values, one for each member of the set s1 over which the equation supply_eq was defined.

Note

By default, all equation attributes introduced above except for .scale are echoed to the solution report that is part of the listing file.

In addition to the equation attributes introduced above, there are a number of equation attributes that cannot be assigned but may be used in computations. They are given in Table 5.

<table>
<thead>
<tr>
<th>Equation Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>.range</td>
<td>The difference between the lower and upper bounds of an equation.</td>
</tr>
<tr>
<td>Slack lower bound</td>
<td>.slacklo</td>
<td>Slack from equation lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of an equation.</td>
</tr>
<tr>
<td>Slack upper bound</td>
<td>.slackup</td>
<td>Slack from equation upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of an equation.</td>
</tr>
<tr>
<td>Slack</td>
<td>.slack</td>
<td>Minimum slack from equation bound. This is defined as the minimum of two values: the slack from equation lower bound and the slack from equation upper bound.</td>
</tr>
<tr>
<td>Infeasibility</td>
<td>.infeas</td>
<td>Amount by which an equation is infeasible falling below its lower bound or above its upper bound. This is defined as max(0, lower bound - level, level - upper bound).</td>
</tr>
</tbody>
</table>

Table 5: Additional Equation Attributes that Cannot Be Assigned but May Be Used in Computations.

4.29.6 Summary and Quick Reference

In this chapter we have covered the declaration and definition of equations in GAMS, arithmetic operations and functions that may be used in equations and equation attributes. A list summarizing the main points to keep in mind follows.
4.30 Model and Solve Statements

4.30.1 Introduction

This chapter brings together all the concepts discussed in previous chapters by explaining how to specify a model and solve it.

4.30.2 The Model Statement

The model statement is used to collect equations into groups and to label them so that they can be solved. The simplest form of the model statement uses the keyword all: the model consists of all equations declared before the model statement is entered. For most simple applications this is all the user needs to know about the model statement.

- Equations must be declared before they may be defined. [1]
- It is good practice to add an explanatory text to the declaration. [2]
- More than one equation may be declared at once. The equation names have to be separated by commas or by a line break. [3]
- Equations may be declared and defined over sets which are called the domain of definition of the equation. [4]
- One indexed equation may generate many constraints depending on the size of the set(s) over which it is defined. [5]
- Equations may be defined over subsets. The dollar condition may be used to filter the members of a set so that only a subset of the members are considered. [6]
- The set(s) over which an equation is defined must be consistent with the set(s) over which the equation was declared, being the set(s) themselves or a subset of the set(s). [7]
- The arrangement of terms in an equation is up to the other. Variables can appear on both sides of an equation. [8]
- Labels of specific set elements may be used explicitly in equations. [9]
- All arithmetic operations that may be used to evaluate expressions are also allowed in equations. [10]
- Many functions that are defined in GAMS may be used in equations. [11]
- It is good practice to set bounds for variables to avoid undefined operations if equations contain operations that are undefined at certain values. [12]
- Equations have attributes similar to variables (.l, .m, .lo, .up and .scale). [13]
4.30.2.1 The Syntax

In general, the syntax for a model declaration in GAMS is as follows:

```
model[s] model_name [text] [ / (all | eqn_name {, eqn_name}) {, var_name(set_name)} /]
{,model_name [text] [ / (all | eqn_name {, eqn_name}) {, var_name(set_name)} /]} ;
```

The keyword `model[s]` indicates that this is a model statement and `model_name` is the internal name of the model in GAMS, it is an identifier. The optional explanatory text is used to describe the model, `all` is a keyword as introduced above and `eqn_name` is the name of an equation that has been declared prior to the model statement. `Var_name(set_name)` is a couple of previous declared variable and and set to limit the domain of variables in the model. More details about this are described in the following subsection. For advice on explanatory text and how to choose a `model_name`, see the tutorial Good Coding Practices.

Note

Model statements for Mixed Complementarity Problem (MCP) and Mathematical Program with Equilibrium Constraints (MPEC) models require a slightly different notation, since complementarity relationships need to be included. For details see subsections Mixed Complementarity Problem (MCP) and Mathematical Program with Equilibrium Constraints (MPEC).

An example of a model definition in GAMS is shown below.

```
Model transport "a transportation model" / all /;
```

The model is called `transport` and the keyword `all` is a shorthand for all known (declared) equations.

Several models may be declared (and defined) in one model statement. This is useful when experimenting with different ways of writing a model, or if one has different models that draw on the same data. Consider the following example, adapted from [PROLOG], in which different groups of the equations are used in alternative versions of the problem. Three versions are solved: the linear, nonlinear, and 'expenditure' versions. The model statement to define all three is:

```
Model nortonl "linear version" / cb,rc,dfl,bc,obj /
nortonn "nonlinear version" / cb,rc,dfn,bc,obj /
nortone "expenditure version" / cb,rc,dfe,bc,obj /;
```

Here `cb`, `rc`, etc. are the names of the equations. We will describe below how to obtain the solution to each of the three models.

Note

If several models are declared and defined with one model statement, the models have to be separated by commas or linefeeds and a semicolon terminates the entire statement.

If several models are declared then it is possible to use one previously declared model in the declaration of another. The following examples illustrate this:

```
Model one "first model" / tcost_eq, supply_eq, demand_eq /
two "second model that nests first" / one, balance_eq /
three "third model that nests first and second" / two, capacity_eq, configure_eq /;
```
Model one is declared and defined using the general syntax, model two contains all the equations of model one and the equation balance_eq, and model three contains all of model two and the equations capacity_eq and configure_eq.

In addition to nesting models as illustrated above, it is also possible to use the symbols + and - to augment or remove items relative to models that were previously defined. The following examples serve as illustration:

Model four "fourth model: model three minus model one" / three-one /  
  five "fifth model: model three without eqn configure_eq" / three-configure_eq /  
  six "sixth model: model four plus model two" / four+two /;

Model four contains the equations from model three except for those that belong to model one. Model five contains all equations from model three except for equation configure_eq. Model six contains the union of the equations in model four and two. Note that both model names and equation names may be used in association with the symbols + and -.

4.30.2.1.1 Limited domain for variables  As mentioned above, it is possible to limit the domain of variables used in a model in the model statement. This allows to restrict the generation of blocks of variables in a single place instead of using, e.g., dollar conditions at every place where this variable block is used in equations (which might be required for an efficient model generation).

The following examples is based on the basic transportation model [TRNSPORT]. To limit the transportation network in that model to certain links (e.g. because some are blocked because of some reason) one could introduce a subset of the free links and use that with dollar conditions in the equations like this:

* Initialize whole network as free  
Set freeLinks(i,j) Useable links in the network / #i.#j /;

  cost.. z =e= sum((i,j), c(i,j)*x(i,j)$freeLinks(i,j));
  supply(i).. sum(j, x(i,j)$freeLinks(i,j)) =l= a(i);  
  demand(j).. sum(i, x(i,j)$freeLinks(i,j)) =g= b(j);

* Block a particular link  
freetLinks('san-diego','topeka') = no;

Model transport / all /;  
solve transport using lp minimizing z;

Now, instead of adding the dollar condition to each appearance of x in the model, one could simply add a domain restriction for that variable to the model statement directly by specifying a variable and the set that limits its domain. Using this approach, the previous example looks like this:

* Initialize whole network as free  
Set freeLinks(i,j) Useable links in the network / #i.#j /;

  cost.. z =e= sum((i,j), c(i,j)*x(i,j));
  supply(i).. sum(j, x(i,j)) =l= a(i);  
  demand(j).. sum(i, x(i,j)) =g= b(j);

* Block a particular link  
freetLinks('san-diego','topeka') = no;

Model transport / all, x(freeLinks) /;  
solve transport using lp minimizing z;
Note

If one adds the domain restriction to the model statement, internally GAMS inserts a dollar condition to every appearance of the restricted variables in equations of the model. When doing this, the indices are copied as they appear with the variable. So, in the example above, \( x(i,j) \) becomes \( x(i,j)\$freeLinks(i,j) \). In the same way \( x(i-1,j+1) \) becomes \( x(i-1,j+1)\$freeLinks(i-1,j+1) \) and \( x('seattle','chicago') \) becomes \( x('seattle','chicago)\$freeLinks('seattle','chicago') \).

Attention

As a consequence of above's note one could see some unexpected results, like "division by zero errors", if it is not done carefully. For example, the following dummy model, will trigger such an error, since we sum over all \( i \), but some \( x \) were excluded leaving a 0 as divisor:

```plaintext
Set i / i1*i3 /
   sub(i) / i2 /;
Positive Variable x(i);
Variable     z;
Equation     obj;

obj.. z =e= sum(i, 1/x(i));
x.lo(i) = 1;
Model m / obj, x(sub) /;
solve m min z use nlp;
```

4.30.2.2 Classification of Models

Various types of problems can be solved with GAMS. Note that the type of the model must be known before it may be solved. The model types are briefly discussed in this section. GAMS checks that the model is in fact the type the user thinks it is, and issues explanatory error messages if it discovers a mismatch - for instance, that a supposedly linear model contains nonlinear terms. Some problems may be solved in more than one way, and the user has to choose which way to use. For instance, if there are binary or integer variables in the model, it can be solved either as a MIP or as a RMIP.

The model types and their identifiers, which are needed in the a solve statement, are given in Table 1. For details on the solve statement, see section The Solve Statement.

<table>
<thead>
<tr>
<th>GAMS Model Type</th>
<th>Model Type Description</th>
<th>Requirements and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>Linear Program</td>
<td>Model with no nonlinear terms or discrete (i.e. binary, integer, etc) variables.</td>
</tr>
<tr>
<td>NLP</td>
<td>Nonlinear Program</td>
<td>Model with general nonlinear terms involving only smooth functions, but no discrete variables. For a classification of functions as to smoothness, see section Functions.</td>
</tr>
<tr>
<td>QCP</td>
<td>Quadratically Constrained Program</td>
<td>Model with linear and quadratic terms, but no general nonlinear terms or discrete variables.</td>
</tr>
<tr>
<td>DNLP</td>
<td>Discontinuous Nonlinear Program</td>
<td>Model with non-smooth nonlinear terms with discontinuous derivatives, but no discrete variables. This is the same as NLP, except that non-smooth functions may appear as well. These models are more difficult to solve than normal NLP models and we strongly advise not to use this model type.</td>
</tr>
<tr>
<td>GAMS Model Type</td>
<td>Model Type Description</td>
<td>Requirements and Comments</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed Integer Program</td>
<td>Model with binary, integer, SOS and/or semi variables, but no nonlinear terms.</td>
</tr>
<tr>
<td>RMIP</td>
<td>Relaxed Mixed Integer Program</td>
<td>Like MIP, except that the discrete variable requirement is relaxed. See the note below on relaxed model types.</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed Integer Nonlinear Program</td>
<td>Model with both nonlinear terms and discrete variables.</td>
</tr>
<tr>
<td>RMINLP</td>
<td>Relaxed Mixed Integer Nonlinear Program</td>
<td>Like MINLP except that the discrete variable requirement is relaxed. See the note below on relaxed model types.</td>
</tr>
<tr>
<td>MIQCP</td>
<td>Mixed Integer Quadratically Constrained Program</td>
<td>Model with both quadratic terms and discrete variables, but no general nonlinear term.</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>Relaxed Mixed Integer Quadratically Constrained Program</td>
<td>Like MIQCP except that the discrete variable requirement is relaxed. See the note below on relaxed model types.</td>
</tr>
<tr>
<td>MCP</td>
<td>Mixed Complementarity Problem</td>
<td>A square, possibly nonlinear, model that generalizes a system of equations. Rows and columns are matched in one-to-one complementary relationships.</td>
</tr>
<tr>
<td>CNS</td>
<td>Constrained Nonlinear System</td>
<td>Model solving a square, possibly nonlinear system of equations, with an equal number of variables and constraints.</td>
</tr>
<tr>
<td>MPEC</td>
<td>Mathematical Programs with Equilibrium Constraints</td>
<td>A difficult model type for which solvers and reformulations are currently being developed.</td>
</tr>
<tr>
<td>RMPEC</td>
<td>Relaxed Mathematical Program with Equilibrium Constraints</td>
<td>A difficult model type for which solvers and reformulations are currently being developed. See the note below on relaxed model types.</td>
</tr>
<tr>
<td>EMP</td>
<td>Extended Mathematical Program</td>
<td>A family of mathematical programming extensions.</td>
</tr>
<tr>
<td>MPSGE</td>
<td>General Equilibrium</td>
<td>Not actually a model type but mentioned for completeness, see MPSGE.</td>
</tr>
</tbody>
</table>

**Table 1: GAMS Model Types**

**Note**

- The relaxed model types RMIP, RMINLP, RMIQCP, and RMPEC solve the problem as the corresponding model type (e.g. MIP for RMIP) but relax the discrete requirement of the discrete variables. This means that integer and binary variables may assume any values between their bounds. SemiInteger and SemiCont variables may assume any values between 0 and their upper bound. For SOS1 and SOS2 variables the restriction of the number of non-zero values is removed.

- Many "LP" solvers like Cplex offer the functionality of solving convex quadratic models. So the Q matrices in the model need to be positive semidefinite. An extension to this are the
second-order cone programs (SOCP) with either symmetric or rotated cones. See the solver manuals (e.g. on MOSEK) for details.

• Unlike other checks on the model algebra (e.g. existence of discrete variables or general non-linear terms), the GAMS compiler does not enforce a quadratic model to only consist of quadratic and linear terms. This requirement is enforced at runtime for a particular model instance.

4.30.2.2.1 Linear Programming (LP) Mathematically, the Linear Programming (LP) problem looks like:

\[
\begin{align*}
\text{Minimize or maximize} & \quad cx \\
\text{subject to} & \quad Ax \alpha b \\
& \quad L \leq x \leq U,
\end{align*}
\]

where \(x\) is a vector of variables that are continuous real numbers, \(cx\) is the objective function, and \(Ax \alpha b\) represents the set of constraints. Here, \(\alpha\) is an equation operator. For details on the equation types allowed in GAMS, see Equation Types. \(L\) and \(U\) are vectors of lower and upper bounds on the variables.

GAMS supports free (unrestricted) variables, positive variables, and negative variables. Note that users may customize lower and upper bounds, for details see section Bounds on Variables.

For information on LP solvers that can be used through GAMS see the Solver/Model type Matrix.

4.30.2.2 Nonlinear Programming (NLP) Mathematically, the Nonlinear Programming (NLP) problem looks like:

\[
\begin{align*}
\text{Minimize or Maximize} & \quad f(x) \\
\text{subject to} & \quad g(x) \alpha 0 \\
& \quad L \leq x \leq U,
\end{align*}
\]

where \(x\) is a vector of variables that are continuous real numbers, \(f(x)\) is the objective function, and \(g(x) \alpha 0\) represents the set of constraints. For details on the equation types allowed in GAMS, see Equation Types. Note that the functions \(f(x)\) and \(g(x)\) have to be differentiable. \(L\) and \(U\) are vectors of lower and upper bounds on the variables.

For information on NLP solvers that can be used through GAMS see the Solver/Model type Matrix. See also the tutorial Good NLP Formulations.

Note

NLP models may have the nonlinear terms inactive. In this case setting the model attribute TryLinear to 1 causes GAMS to check the model and use the default LP solver if possible. For details on model attributes, see subsection Model Attributes.
4.30.2.2.3 Quadratically Constrained Programs (QCP)  Mathematically, the Quadratically Constrained Programming (QCP) problem looks like:

Maximize or Minimize \( cx + x'Qx \)
subject to \( A_i x + x'R_i x \leq b_i \) for all \( i \)
\( L \leq x \leq U, \)

where \( x \) denotes a vector of variables that are continuous real numbers, \( cx \) is the linear part of the objective function, \( x'Qx \) is the quadratic part of the objective function, \( A_i x \) represents the linear part of the \( i \)th constraint, \( x'R_i x \) its quadratic part and \( b_i \) its right-hand side. For details on the equation types allowed in GAMS, see Equation Types”. Further, \( L \) and \( U \) are vectors of lower and upper bounds on the variables.

Note that a QCP is a special case of the NLP in which all the nonlinearities are required to be quadratic. As such, any QCP model can also be solved as an NLP. However, most "LP" vendors provide routines to solve LP models with a quadratic objective. Some allow quadratic constraints as well. Solving a model using the QCP model type allows these "LP" solvers to be used to solve quadratic models as well as linear ones. Some NLP solvers may also take advantage of the special (quadratic) form when solving QCP models.

Attention

In case a model with quadratic constraints is passed to a QCP solver that only allows a quadratic objective, a capability error will be returned (solver status 6 CAPABILITY PROBLEMS). Some solvers will fail when asked to solve a non-convex quadratic problems as described above.

Note

Using the model attribute TryLinear causes GAMS to see if the problem can be solved as an LP problem. For details on model attributes, see subsection Model Attributes.

For information on QCP solvers that can be used through GAMS see the Solver/Model type Matrix.

4.30.2.2.4 Nonlinear Programming with Discontinuous Derivatives (DNLP)  Mathematically, the Nonlinear Programming with Discontinuous Derivatives (DNLP) problem looks like:

Maximize or Minimize \( f(x) \)
subject to \( g(x) \leq 0 \)
\( L \leq x \leq U, \)

where \( x \) is a vector of variables that are continuous real numbers, \( f(x) \) is the objective function, \( g(x) \leq 0 \) represents the set of constraints, and \( L \) and \( U \) are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see Equation Types. Note that this is the same as NLP, except that non-smooth functions, like \( \text{abs}, \min, \max \) may appear in \( f(x) \) and \( g(x) \).

For information on DNLP solvers that can be used through GAMS see the Solver/Model type Matrix.

Attention

- We strongly advise against using the model type DNLP. The best way to model discontinuous functions is with binary variables, which results in a model of the type MINLP. The model [ABSMMIP] demonstrates this formulation technique for the functions \( \text{abs}, \min, \max \) and \( \text{sign} \). See also section Reformulating DNLP Models.
- Solvers may have difficulties when dealing with the discontinuities, since they are really NLP solvers and the optimality conditions and the reliance on derivatives may be problematic. Using a global solver may alleviate this problem.
4.30.2.2.5 Mixed Integer Programming (MIP) Mathematically, the Mixed Integer Linear Programming (MIP) problem looks like:

Maximize or Minimize \[ c_1 t + c_2 u + c_3 v + c_4 w + c_5 x + c_6 y + c_7 z \]
subject to \[ A_1 t + A_2 u + A_3 v + A_4 w + A_5 x + A_6 y + A_7 z \leq b \]
\[ t \in \mathbb{R} \]
\[ u \geq 0 \quad \text{and} \quad u \leq L_2 \quad \text{and} \quad u \in \mathbb{Z} \]
\[ v \in (0, 1) \]
\[ w \in \text{SOS1} \]
\[ x \in \text{SOS2} \]
\[ y = 0 \quad \text{or} \quad L_6 \leq y \]
\[ z = 0 \quad \text{or} \quad L_7 \leq z \quad \text{and} \quad z \in \mathbb{Z}, \]

where

- \( c_1 t + c_2 u + c_3 v + c_4 w + c_5 x + c_6 y + c_7 z \) is the objective function,
- \( A_1 t + A_2 u + A_3 v + A_4 w + A_5 x + A_6 y + A_7 z \leq b \) represents the set of constraints of various equality and inequality forms,
- \( t \) is a vector of variables that are continuous real numbers,
- \( u \) is a vector of variables that can only take integer values smaller than \( L_2 \),
- \( v \) is a vector of binary variables,
- \( w \) is a vector of variables that belong to \( \text{SOS1} \) sets; this means that at most one variable in the set is nonzero,
- \( x \) is a vector of variables that belong to \( \text{SOS2} \) sets; this means that at most two adjacent variables in the set are nonzero,
- \( y \) is a vector of variables that are semi-continuous; they are either zero or larger than \( L_6 \),
- \( z \) is a vector of variables that are semi-integer; they are integer and either zero or larger than \( L_7 \).

For details on the equation types allowed in GAMS, see Equation Types. For more details on MIPs in GAMS, especially the use of \( \text{SOS} \) and \( \text{semi} \) variables, see section Special Mixed Integer Programming (MIP) Features.

For information on MIP solvers that can be used through GAMS see the Solver/Model type Matrix.

Attention

Not all MIP solvers cover all the cases associated with \( \text{SOS} \) and \( \text{semi} \) variables. Please consult the solver manuals for details on capabilities.

4.30.2.2.6 Mixed Integer Nonlinear Programming (MINLP) Mathematically, the Mixed Integer Nonlinear Programming (MINLP) problem looks like:

Maximize or Minimize \[ f(x) + Dy \]
subject to \[ g(x) + Hy \leq 0 \]
\[ L \leq x \leq U \]
\[ y = \{0, 1, 2, \ldots\}, \]

where \( x \) is a vector of variables that are continuous real numbers, \( y \) denotes a vector of variables that can only take integer values, \( f(x) + Dy \) is the objective function, \( g(x) + Hy \leq 0 \) represents the set of constraints, and \( L \) and \( U \) are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see Equation Types. Further, \( y = \{0, 1, 2, \ldots\} \) is the integrality restriction on \( y \).

For information on MINLP solvers that can be used through GAMS see the Solver/Model type Matrix.
4.30 Model and Solve Statements

Note

• SOS and semi variables can also be accommodated by some solvers. Please consult the solver manuals for details on capabilities.

• The model attribute TryLinear causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection Model Attributes.

4.30.2.2.7 Mixed Integer Quadratically Constrained Programs (MIQCP) A Mixed Integer Quadratically Constrained Program (MIQCP) is a special case of the MINLP in which all the nonlinearities are required to be quadratic. For details see the description of the QCP, a special case of the NLP.

For information on MIQCP solvers that can be used through GAMS, see the Solver/Model type Matrix.

Note

The model attribute TryLinear causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection Model Attributes.

4.30.2.2.8 Mixed Complementarity Problem (MCP) Unlike the other model types we have introduced so far, the Mixed Complementarity Problem (MCP) does not have an objective function. An MCP is specified by three pieces of data: a function \( F(z) : \mathbb{R}^n \rightarrow \mathbb{R}^n \), lower bounds \( \ell \in \{ \mathbb{R} \cup \{-\infty\} \}^n \) and upper bounds \( u \in \{ \mathbb{R} \cup \{\infty\} \}^n \). A solution is a vector \( z \in \mathbb{R}^n \) such that for each \( i \in \{1, \ldots, n\} \), one of the following three conditions hold:

\[
\begin{align*}
F_i(z) = 0 & \quad \text{and} \quad \ell_i \leq z_i \leq u_i \quad \text{or} \\
F_i(z) > 0 & \quad \text{and} \quad z_i = \ell_i \quad \text{or} \\
F_i(z) < 0 & \quad \text{and} \quad z_i = u_i.
\end{align*}
\]

This problem can be written compactly as

\[
F(z) \perp L \leq z \leq U,
\]

where the symbol \( \perp \) (which means "perpendicular to", shortened to "perp to") indicates pair-wise complementarity between the function \( F \) and the variable \( z \) and its bounds.

The following special case is an important and illustrative example:

\[
F(z) \perp z \geq 0.
\]

In this example, the unstated but implied upper bound \( u \) is infinity. Since \( z \) is finite, we cannot have \( z_i = u_i \) and the third condition above cannot hold: this implies \( F(z) > 0 \). The remaining two conditions imply pair-wise complementarity between \( z \geq 0 \) and \( F(z) \geq 0 \). This is exactly the Nonlinear Complementarity Problem, often written as

\[
F(z) \geq 0, \quad z \geq 0, \quad \langle F(z), z \rangle = 0.
\]

None of this rules out the degenerate case (i.e. \( F_i(z) \) and \( z_i \) both zero). In practice, these can be difficult models to solve.

Another special case arises when the bounds \( L \) and \( U \) are infinite. In this case, the second and third conditions above cannot hold, so we are left with \( F(z) = 0 \), a square system of nonlinear equations. And finally, we should mention a special case that occurs frequently in practice: if \( \ell_i = u_i \) (i.e. \( z_i \) is fixed) then we have a complementary pair: one of the three conditions will hold as long as \( F_i(z) \) is defined. Essentially, fixing a variable removes or obviates the matching equation. This is often useful when modeling with MCP.

The definition above describes the canonical MCP model as it exists when GAMS passes it to an MCP solver. Some models have exactly this form even in the GAMS code, but usually some processing is done by the GAMS system to arrive at a model in this form. Here we'll describe the steps of this process and illustrate with an example from the model library.
1. The process starts with the list of rows (aka single equations) and columns (aka single variables) that make up the MCP model, and potentially some matching information.

- The usual rules apply: rows are part of the model because their associated equations are included in the model statement, but columns only become part of the model by use: a column enters the model only if it is used in some row of the model. Therefore including a variable symbol as part of a match in the model statement will not influence the set of columns belonging to the model.
- Matches (where they exist) are pointers from rows to columns.
- Technically, the MCP is defined via a function $F$ while a model contains constraints. Given a constraint, we define an associated function as LHS - RHS, so e.g. $F_i \geq 0$ is consistent with a =G= constraint.

2. The explicit matches are processed: each match creates a complementary pair. What remains after the explicit matches are consumed are the unmatched rows and unmatched columns.

- It is an error for any column to be matched to multiple rows, so the row-column matching is one-to-one.
- For each match some consistency checks between the column bounds and the row type are made. For details, see Table 2.
- For example, matching an =N= row with any column is good, matching an =E= row with a free column is good, matching an =E= row with a lower-bounded column is allowed, and matching a =G= row with an upper-bounded column results in an error.

3. Any fixed columns remaining are ignored: these columns can be treated like exogenous variables or parameters.

4. If what remains is a set of =E= rows and an equal number of unbounded columns, these can be matched up in any order and we have a well-defined MCP. If this is not what remains, an error is triggered.

To illustrate how this works, consider the spatial equilibrium model [SPATEQU] with the following model statement:

```
Model P2R3_MCP / dem, sup, in_out.p, dom_trad.x /;
```

1. The model P2R3_MCP includes the rows from equations dem, sup, in_out and exactly the columns used by these rows. Checking the listing file, we see columns for Qd, Qs, x, and p. In addition, the model statement specifies two matches: in_out.p and dom_trad.x. These matches always take the form of an equation.variable pair, with no indices or domains included.

2. In this example, the rows corresponding to the equation in_out match up perfectly with the columns from the variable p: there are no holes in the set of rows or columns because of some dollar conditions in the equation definition. We have a one-to-one match so all the rows of in_out and columns of p are consumed by the match in the model statement. The same holds for the dom_trad.x pair, so what is left are the rows of dem and sup and the columns of Qd and Qs, all of which are unmatched.

3. There are no fixed variables to remove.

4. Since dem and sup are =E= constraints and Qd and Qs are free variables, we can match them in any order without changing the solution set for this model. The counts of these unmatched equality rows and unmatched free variables are equal, so we get a well-defined MCP.

When rows are matched explicitly to columns, some care must be taken to match them consistently. For example, consider a row-column match g.y. The row g can be of several types: =N=, =E=, =G=, or =L=. An =N= row can be matched to any sort of variable: the =N= doesn’t imply any sort of relationship, which works perfectly with our definition of $\perp$ above: the allowed sign or direction of $g$ is determined completely...
by the bounds on the complementary variable $y$. If $g$ is an $=E=$ row, this is consistent with a free variable $y$, but what if $y$ has an active lower bound? By definition we allow $g$ to be positive at solution, but this violates the declaration as an $=E=$ row. Such cases can be handled by marking the row with a *redef*. The total number of redefs for a given model is available via the `NumRedef` model attribute and is shown in the report summary. Note that the set of rows marked depends on the solution: in the example above, if $g$ is zero at solution it will not be marked as a redef, regardless of what the bounds are on $y$. Finally, some combinations are simply not allowed: they will result in a model generation error. The table below lists the outcome for all possible combinations.

**Table 2: MCP Matching**

<table>
<thead>
<tr>
<th>Column Bounds</th>
<th>$=N=$</th>
<th>$=E=$</th>
<th>$=G=$</th>
<th>$=L=$</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower</td>
<td>OK</td>
<td>redef</td>
<td>OK</td>
<td>ERROR</td>
</tr>
<tr>
<td>upper</td>
<td>OK</td>
<td>redef</td>
<td>ERROR</td>
<td>OK</td>
</tr>
<tr>
<td>free</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>double</td>
<td>OK</td>
<td>redef</td>
<td>redef</td>
<td>redef</td>
</tr>
<tr>
<td>fixed</td>
<td>OK</td>
<td>redef</td>
<td>redef</td>
<td>redef</td>
</tr>
</tbody>
</table>

The definition, process, and rules above have several implications for valid MCP models:

- It is always acceptable to use the $=N=$ notation when defining the equations in an MCP model, provided these equations are matched explicitly. In this case the bounds on $F(z)$ are implied by the bounds on the matching columns, and redefs will never occur.
- Variables that are known to be lower-bounded (no upper bound) will match consistently with $=G=$ equations.
- Variables that are known to be upper-bounded (no lower bound) will match consistently with $=L=$ equations.
- Variables that are known to be unbounded will match consistently with $=E=$ equations.
- Where the bound structure is not known in advance, or both upper and lower bounds exist, a match with an $=N=$ equation will always be consistent. Other equation types will result in errors or redefs.
- The model may initially have fewer rows than columns, as long as the “extra” columns are unmatched fixed columns that ultimately get removed from the MCP passed to the solver.
- Any bounded-but-not-fixed column must be matched explicitly to a row.
- The only rows that may be unmatched are $=E=$ rows.
- It is customary to re-use the constraints of an LP or NLP model when formulating the MCP corresponding to the Karush-Kuhn-Tucker (KKT) conditions. If the original model is a *minimization*, the LP/NLP marginals .m and the variables for these marginals in the MCP will use the same sign convention, and the orientation for the constraints will be consistent between the two models, making re-use easier.

As mentioned above, it is typical to use the same equations in both NLP and MCP models. Sometimes, it is not the original equation that is wanted for the MCP, but rather the reoriented (aka negated or flipped) equation. For example, the flipped version of $x^{++1.5} =L= y$ is $y =G= x^{++1.5}$, while $\text{sqr}(u) - \text{sqr}(v) =E= 5$ becomes $-\text{sqr}(u) + \text{sqr}(v) =E= -5$. Instead of re-implementing the equation in flipped form, the same result can be achieved by prefixing the equation name with a `-` in the model statement. See the [mcp10] model for an example of such usage. When equations are used in flipped form, they are marked with a *redir* in the listing file’s solution listing.

An example of complementarity that should be familiar to many is the relationship between a constraint and its associated dual multiplier: if the constraint is non-binding, its dual multiplier must be zero (i.e. at
bound) while if a dual multiplier is nonzero the associated constraint must be binding. In fact, the KKT or first-order optimality conditions for LP and NLP models can be expressed and solved as an MCP.

These complementarity relationships found in optimization problems are useful in understanding the marginal values assigned to rows and columns in the GAMS solution for MCP. With no objective function, the usual definition for marginal values and their interpretation isn’t useful. Instead, the GAMS MCP convention for the marginal values of columns is to return the slack of the associated row (i.e. its value when interpreted and evaluated as a function). For the marginal values of rows, the level value (not the slack) of the associated column is returned. When we apply this convention to the NCP \( F(z) \geq 0, z \geq 0, \langle F(z), z \rangle = 0 \) we see pairwise complementarity between the levels and marginals returned for each of the rows and columns in the model. This is also the case if we take the KKT conditions of an LP in a suitable standard form: minimization, \( x \geq 0, Ax \geq b \).

MCPs arise in many application areas including applied economics, game theory, structural engineering and chemical engineering. For further details on this class of problems, see http://www.neos-guide.org/content/complementarity-problems.

For information on MCP solvers that can be used through GAMS, see Solver/Model type Matrix.

### 4.30.2.2.9 Constrained Nonlinear System (CNS)

The Constrained Nonlinear System (CNS) is the second GAMS model type that does not have an objective function. Mathematically, a CNS model looks like:

\[
\begin{align*}
\text{Find} \\
\text{subject to}
\end{align*}
\]

\[
x
\]

\[
F(x) = 0
\]

\[
L \leq x \leq U
\]

\[
G(x) \alpha b,
\]

where \( x \) is a set of continuous variables and \( F \) is a set of nonlinear equations of the same dimension as \( x \). This is a key property of this model type: the number of equations equals the number of variables, so we have a square system. The (possibly empty) constraints \( L \leq x \leq U \) are not intended to be binding at the solution, but instead are included to constrain the solution to a particular domain, to avoid regions where \( F(x) \) is undefined, or perhaps just to give the solver a push in the right direction. The (possibly empty) constraints \( G(x) \alpha b \) are intended to serve the same purpose as the variable bounds and are silently converted to equations with bounded slacks.

Note that since there is no objective in a CNS model, there are no marginal values for variables and equations. Any marginal values already stored in the GAMS database will remain untouched. CNS models also make use of some model status values that allow a solver to indicate if the solution is unique (e.g. for a non-singular linear system) or if the linearization is singular at the solution. For singular models (solved or otherwise), the solver can mark one or more dependent rows with a \( \text{depnd} \). The total number of rows so marked for a given model is available via the \( \text{NumDepnd} \) model attribute and is shown in the report summary.

The CNS model is a generalization of a square system of equations \( F(x) = 0 \). Such a system could also be modeled as an NLP with a dummy objective. However, there are a number of advantages to using the CNS model type, including:

- A check by GAMS that the model is really square,
- solution/model diagnostics by the solver (e.g. singular at solution, (locally) unique solution),
- CNS-specific warnings if the side constraints \( L \leq x \leq U \) or \( G(x) \alpha b \) are active at a solution,
- and potential improvement in solution times, by taking better advantage of the model properties.

For information on CNS solvers that can be used through GAMS, see the Solver/Model type Matrix.
4.30.2.2.10 Mathematical Program with Equilibrium Constraints (MPEC) Mathematically, the Mathematical Program with Equilibrium Constraints (MPEC) problem looks like:

\[
\begin{align*}
\text{Maximize or Minimize} & \quad f(x, y) \\
\text{subject to} & \quad g(x, y) \leq 0 \\
& \quad L_x \leq x \leq U_x \\
& \quad F(x, y) \perp L_y \leq y \leq U_y,
\end{align*}
\]

where \( x \) and \( y \) are vectors of continuous real variables. The variables \( x \) are often called the control or upper-level variables, while the variables \( y \) are called the state or lower-level variables. \( f(x, y) \) is the objective function. \( g(x, y) \leq 0 \) represents the set of traditional (i.e., NLP-type) constraints; some solvers may require that these constraints only involve the control variables \( x \). The function \( F(x, y) \) and the bounds \( L_y \) and \( U_y \) define the equilibrium constraints. If \( x \) is fixed, then \( F(x, y) \) and the bounds \( L_y \) and \( U_y \) define an MCP; the discussion of the "perp to" symbol \( \perp \) in that section applies here as well. From this definition, we see that the MPEC model type contains NLP and MCP models as special cases of MPEC.

A simple example of an entire MPEC model is given below.

```gams
variable z, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up = 1;
equations cost, g, h1, h2;
cost.. z =E= x1 + x2;
g.. sqr(x1) + sqr(x2) =L= 1;
h1.. x1 =G= y1 - y2 + 1;
h2.. x2 + y2 =N= 0;
model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min z;
```

Note that as in the MCP, the complementarity relationships in an MPEC are specified in the model statement via equation-variable pairs: the \( h1.y1 \) specifies that the equation \( h1 \) is perpendicular to the variable \( y1 \) and the \( h2.y2 \) specifies that the equation \( h2 \) is perpendicular to the variable \( y2 \). For details on the solve statement, see section The Solve Statement.

While the MPEC model formulation is very general, it also results in problems that can be very difficult to solve. The state-of-the-art for MPEC solvers is not nearly as advanced as that for other model types. As a result, you should expect the MPEC solvers to be more limited by problem size and/or robustness issues than solvers for other model types.

For information on MPEC solvers that can be used through GAMS, see the Solver/Model type Matrix. For more details on MPECs and solver development, see [http://www.neos-guide.org/content/mathematical-programs-equilibrium-constraints](http://www.neos-guide.org/content/mathematical-programs-equilibrium-constraints).

4.30.2.2.11 Extended Mathematical Programs (EMP) Extended Mathematical Programming (EMP) is an (experimental) framework for automated mathematical programming reformulations. Using EMP, model formulations that GAMS cannot currently handle directly or for which no robust and mature solver technology exists can be automatically and reliably reformulated or transformed into models for which robust and mature solver technology does exist within the GAMS system. For more details, see the chapter on EMP. Currently EMP supports:

- Equilibrium problems including variational inequalities, Nash games, and Multiple Optimization Problems with Equilibrium Constraints (MOPECs).
Hierarchical optimization problems such as bilevel programs.

- Disjunctive programs for modeling discrete choices with binary variables.

- Stochastic programs including two-stage and multi-stage stochastic programs, chance constraints and risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR).

Apart from the disjunctive and stochastic programming models mentioned above, EMP models are typically processed (aka solved) via the JAMS solver: this solver does the work of reformulation/transformation, calling GAMS to solve this reformulation, and post-processing the solution that results to bring it back in terms of the original EMP model.

Examples demonstrating how to use the EMP framework and the JAMS and DE solvers are available in the GAMS EMP Library. These solvers require no license of their own to run but can and do call subsolvers that do require a license.

4.30.2.3 Model Attributes

Models have attributes that hold a variety of information, including

- information about the results of a solve performed, a solve statement, the solution of a model,
- information about certain features to be used by GAMS or the solver,
- information passed to GAMS or the solver specifying various settings that are also subject to option statements.

Model attributes are accessed in the following way:

\[
\text{model\_name.attribute}
\]

Here model\_name is the name of the model in GAMS and .attribute is the specific attribute that is to be accessed. Model attributes may be used on the left-hand side and the right-hand side of assignments. Consider the following example:

\[
\begin{align*}
\text{transport.resLim} & = 600; \\
x & = \text{transport.modelStat};
\end{align*}
\]

In the first line the attribute .resLim of the model transport is specified to be 600 (seconds). In the second line the value of the attribute .modelStat of the model transport is assigned to the scalar x. Note that model attributes may also be used in display statements.

Some of the attributes are mainly used before the solve statement to provide information to GAMS or the solver link. Others are set by GAMS or the solver link and hence are mainly used after a solve statement.

Moreover, some of the attributes used before the solve may also be set via an option statement or the command line. Consider the following example:

\[
\text{option ResLim=10;}
\]
This line is an option statement and applies to all models. One can set the model attribute .ResLim to overwrite the global ResLim option. In order to revert the individual .ResLim to the global ResLim option, one needs to set the model attribute to NA. For more on option statements, see chapter The Option Statement.

```plaintext
  gams mymodel ResLim=10
```

This sets the global ResLim option when invoking the gams run (e.g. from the command line). For more on command line parameters, see chapter The GAMS Call and Command Line Parameters.

Note that a model-specific option takes precedence over the global setting specified with an option statement and that a setting via an option statement takes precedence over a setting via the command line parameter.

The complete list of model attributes is given below. Observe that each entry is linked to a detailed description of the respective attribute, including information of whether the attribute is also available as command line parameter or option statement. Note that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section Detailed Descriptions of All Options.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRatio</td>
<td>Basis detection threshold</td>
</tr>
<tr>
<td>cheat</td>
<td>Cheat value, i.e. minimum solution improvement threshold</td>
</tr>
<tr>
<td>cutOff</td>
<td>Cutoff value for branch and bound</td>
</tr>
<tr>
<td>defPoint</td>
<td>Indicator for passing on default point</td>
</tr>
<tr>
<td>dictFile</td>
<td>Force writing of a dictionary file if dictfile &gt; 0</td>
</tr>
<tr>
<td>domLim</td>
<td>Domain violation limit solver default</td>
</tr>
<tr>
<td>fdDelta</td>
<td>Step size for finite differences</td>
</tr>
<tr>
<td>fdOpt</td>
<td>Options for finite differences</td>
</tr>
<tr>
<td>holdFixed</td>
<td>Treat fixed variables as constants</td>
</tr>
<tr>
<td>integer1..5</td>
<td>Integer communication cells</td>
</tr>
<tr>
<td>iterLim</td>
<td>Iteration limit of solver</td>
</tr>
<tr>
<td>limCol</td>
<td>Maximum number of columns listed in one variable block</td>
</tr>
<tr>
<td>limRow</td>
<td>Maximum number of rows listed in one equation block</td>
</tr>
<tr>
<td>MCPRHoldFx</td>
<td>Print list of rows that are perpendicular to variables</td>
</tr>
<tr>
<td></td>
<td>removed due to the holdfixed setting</td>
</tr>
<tr>
<td>nodLim</td>
<td>Node limit in branch and bound tree</td>
</tr>
<tr>
<td>optCA</td>
<td>Absolute Optimality criterion solver default</td>
</tr>
<tr>
<td>optCR</td>
<td>Relative Optimality criterion solver default</td>
</tr>
<tr>
<td>optFile</td>
<td>Default option file</td>
</tr>
<tr>
<td>priorOpt</td>
<td>Priority option for variable attribute .prior</td>
</tr>
<tr>
<td>real1..5</td>
<td>Real communication cells</td>
</tr>
<tr>
<td>reform</td>
<td>Reformulation level</td>
</tr>
<tr>
<td>resLim</td>
<td>Wall-clock time limit for solver</td>
</tr>
<tr>
<td>savePoint</td>
<td>Save solver point in GDX file</td>
</tr>
<tr>
<td>scaleOpt</td>
<td>Employ user specified variable and equation scaling</td>
</tr>
<tr>
<td></td>
<td>factors</td>
</tr>
<tr>
<td>solPrint</td>
<td>Solution report print option</td>
</tr>
<tr>
<td>solveLink</td>
<td>Solver link option</td>
</tr>
<tr>
<td>solveOpt</td>
<td>Multiple solve management</td>
</tr>
<tr>
<td>sysOut</td>
<td>Solver Status file reporting option</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads to be used by a solver</td>
</tr>
<tr>
<td>tolInfeas</td>
<td>Infeasibility tolerance for an empty row of the form a.</td>
</tr>
<tr>
<td></td>
<td>0*x =e= 0.0001;</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tolInfRep</td>
<td>This attribute sets the tolerance for marking infeasible in the equation listing</td>
</tr>
<tr>
<td>tolProj</td>
<td>Tolerance for setting solution values to a nearby bound when reading a solution</td>
</tr>
<tr>
<td>tryInt</td>
<td>Whether solver should make use of a partial integer-feasible solution</td>
</tr>
<tr>
<td>tryLinear</td>
<td>Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used</td>
</tr>
<tr>
<td>workFactor</td>
<td>Memory Estimate multiplier for some solvers</td>
</tr>
<tr>
<td>workSpace</td>
<td>Work space for some solvers in MB</td>
</tr>
</tbody>
</table>

### 4.30.2.3.1 Model Attributes Mainly Used Before Solve

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domUsd</td>
<td>Number of domain violations</td>
</tr>
<tr>
<td>etAlg</td>
<td>Solver dependent timing information</td>
</tr>
<tr>
<td>etSolve</td>
<td>Elapsed time it took to execute a solve statement in total</td>
</tr>
<tr>
<td>etSolver</td>
<td>Elapsed time taken by the solver only</td>
</tr>
<tr>
<td>handle</td>
<td>Unique handle number of SOLVE statement</td>
</tr>
<tr>
<td>iterUsd</td>
<td>Number of iterations used</td>
</tr>
<tr>
<td>line</td>
<td>Line number of last solve of the corresponding model</td>
</tr>
<tr>
<td>linkUsed</td>
<td>Integer number that indicates the value of SolveLink used for the last solve</td>
</tr>
<tr>
<td>marginals</td>
<td>Indicator for marginals present</td>
</tr>
<tr>
<td>maxInfes</td>
<td>Maximum of infeasibilities</td>
</tr>
<tr>
<td>meanInfes</td>
<td>Mean of infeasibilities</td>
</tr>
<tr>
<td>modelStat</td>
<td>Integer number that indicates the model status</td>
</tr>
<tr>
<td>nodUsd</td>
<td>Number of nodes used by the MIP solver</td>
</tr>
<tr>
<td>number</td>
<td>Model instance serial number</td>
</tr>
<tr>
<td>numDepnd</td>
<td>Number of dependencies in a CNS model</td>
</tr>
<tr>
<td>numDVar</td>
<td>Number of discrete variables</td>
</tr>
<tr>
<td>numEqu</td>
<td>Number of equations</td>
</tr>
<tr>
<td>numInfes</td>
<td>Number of infeasibilities</td>
</tr>
<tr>
<td>numNLIns</td>
<td>Number of nonlinear instructions</td>
</tr>
<tr>
<td>numNLNZ</td>
<td>Number of nonlinear nonzeros</td>
</tr>
<tr>
<td>numNOpt</td>
<td>Number of nonoptimalities</td>
</tr>
<tr>
<td>numNZ</td>
<td>Number of nonzero entries in the model coefficient matrix</td>
</tr>
<tr>
<td>numRedef</td>
<td>Number of MCP redefinitions</td>
</tr>
<tr>
<td>numVar</td>
<td>Number of variables</td>
</tr>
<tr>
<td>numVarProj</td>
<td>Number of bound projections during model generation</td>
</tr>
<tr>
<td>objEst</td>
<td>Estimate of the best possible solution for a mixed-integer model</td>
</tr>
<tr>
<td>objVal</td>
<td>Objective function value</td>
</tr>
<tr>
<td>procUsed</td>
<td>Integer number that indicates the used model type</td>
</tr>
<tr>
<td>resCalc</td>
<td>Time spent in function and derivative calculations (deprecated)</td>
</tr>
<tr>
<td>resDeriv</td>
<td>Time spent in derivative calculations (deprecated)</td>
</tr>
<tr>
<td>resGen</td>
<td>Time GAMS took to generate the model in CPU seconds (deprecated)</td>
</tr>
<tr>
<td>resIn</td>
<td>Time to import model (deprecated)</td>
</tr>
<tr>
<td>resOut</td>
<td>Time to export solution (deprecated)</td>
</tr>
<tr>
<td>resUsd</td>
<td>Time the solver used to solve the model in seconds</td>
</tr>
<tr>
<td>rngBndMax</td>
<td>Maximum absolute non-zero value of bounds passed to the solver (excluding infinity)</td>
</tr>
<tr>
<td>rngBndMin</td>
<td>Minimum absolute non-zero value of bounds passed to the solver</td>
</tr>
</tbody>
</table>
## 4.30 Model and Solve Statements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rngMatMax</td>
<td>Maximum absolute non-zero value of coefficients in the model matrix passed to the solver (excluding infinity)</td>
</tr>
<tr>
<td>rngMatMin</td>
<td>Minimum absolute non-zero value of coefficients in the model matrix passed to the solver</td>
</tr>
<tr>
<td>rngRhsMax</td>
<td>Maximum absolute non-zero value of right hand sides passed to the solver (excluding infinity)</td>
</tr>
<tr>
<td>rngRhsMin</td>
<td>Minimum absolute non-zero value of right hand sides passed to the solver</td>
</tr>
<tr>
<td>rObj</td>
<td>Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish</td>
</tr>
<tr>
<td>solveStat</td>
<td>Indicates the solver termination condition</td>
</tr>
<tr>
<td>sumInfes</td>
<td>Sum of infeasibilities</td>
</tr>
<tr>
<td>sysIdent</td>
<td>Solver identification number</td>
</tr>
<tr>
<td>sysVer</td>
<td>Solver version</td>
</tr>
</tbody>
</table>

### 4.30.2.3.2 Model Attributes Mainly Used After Solve

### 4.30.3 The Solve Statement

Once a model has been defined using the model statement, the solve statement prompts GAMS to call one of the available solvers for the particular model type. This section introduces and discusses the solve statement in detail. For a list of GAMS model types, see Table 1. For information on how to specify desired solvers, see section Choosing a Solver.

**Note**

It is important to remember that GAMS does *not* solve the problem, but passes the problem definition to one of a number of separate solver programs that are integrated with the GAMS system.

### 4.30.3.1 The Syntax of the Solve Statement

In general, the syntax for a `solve` statement is as follows. Note that there are two alternatives that are equally valid:

```gams
solve model_name using model_type maximizing|minimizing var_name;
solve model_name maximizing|minimizing var_name using model_type ;
```

The keyword `solve` indicates that this is a `solve` statement. `Model_name` is the name of the model as defined by a `model` statement. Note that the `model` statement must be placed before the `solve` statement in the program. The keyword `using` is followed by `model_type`, which is one of the GAMS model types described above, see Table 1. The keywords `maximizing` or `minimizing` indicate the direction of the optimization. `Var_name` is the name of the objective variable that is being optimized. An example of a `solve` statement in GAMS is shown below.

```gams
Solve transport using lp minimizing cost ;
```

`Solve` and `using` are reserved words, `transport` is the name of the model, `lp` is the model type, `minimizing` is the direction of optimization, and `cost` is the objective variable. Note that an objective variable is used instead of an objective row or function.
Attention

The objective variable must be scalar and of type free, and must appear in at least one of the equations in the model.

Recall that some model types (e.g. the Constrained Nonlinear System (CNS) or the Mixed Complementarity Problem (MCP)) do not have an objective variable. So their solve statement syntax is slightly different:

solve model_name using model_type;

As before, solve and using are keywords, model_name is the name of the model as defined by a model statement and model_type is the GAMS model type CNS or MCP. There is no objective variable and consequently no direction of optimization. An example from the spatial equilibrium model [SPATEQU] illustrates this solve statement:

Solve P2R3_MCP using mcp;

P2R3_MCP is the model name, the model type is MCP and as expected, there is no objective variable.

The EMP model type serves many purposes including some experimental ones. The solve statement with model type EMP can be with or without the objective variable and optimization direction. For more information, see chapter Extended Mathematical Programming (EMP).

4.30.3.2 Actions Triggered by the Solve Statement

When GAMS encounters a solve statement during compilation (the syntactic check of the input file) or execution (actual execution of the program), it initiates a number of special actions. The purpose is to prevent waste that would be caused by solving a model that has apparently been incorrectly specified. During compilation the following are verified:

1. All symbolic equations have been defined and the objective variable is used in at least one of the equations.
2. The objective variable is scalar and of type free (even though lower and upper bounds may have been specified)
3. MCP models are checked for appropriate complementarity and squareness.
4. Each equation fits into the specified problem class (linearity for LP, continuous derivatives for NLP, as outlined above).
5. All sets and parameters in the equations have values assigned.
4.30 Model and Solve Statements

Note

GAMS issues explanatory error messages if it discovers that the model is not according to type; for example, the presence of nonlinear terms in a supposedly LP model. For details on error messages, see chapter GAMS Output.

At execution time the solve statement triggers the following sequence of steps:

1. The model is translated into the representation required by the solution system to be used.

2. Debugging and comprehension aids that the user wishes to see are produced and written to the output file (EQUATION LISTING, etc). For customizing options (e.g. LimRow and LimCol), see chapter The Option Statement.

3. GAMS verifies that there are no inconsistent bounds or unacceptable values (for example, NA or UNDF) in the problem.

4. Any errors detected at this stage cause termination with as much explanation as possible, using the GAMS names for the identifiers causing the trouble.

5. GAMS designs a solution strategy based on the possible availability of level values or basis information from a previous solution: all available information is used to provide efficiency and robustness of operation. Any specifications provided by the user (Iteration limits etc.) are incorporated. A solver is chosen which is either the default solver for that problem type, the solver specified on the command line or the solver chosen by an option statement. For details see section Choosing a Solver.

6. GAMS passes control to the solution subsystem and waits while the problem is being solved.

7. GAMS reports on the status of the solution process and loads solution values back into the GAMS database. This causes new values to be assigned to the .l and .m fields for all individual equations and variables in the model. In addition, the post solution model attributes are assigned. The procedure for loading back the data associated with level and marginal values may be customized using the SolveOpt model attribute and option. A row by row and column by column listing of the solution is provided by default. It may be suppressed by the SolPrint model attribute or option. Any apparent difficulty with the solution process will cause explanatory messages to be displayed. Errors caused by forbidden nonlinear operations are reported at this stage.

Note

When the solver does not provide a dual solution (.m), then GAMS does not print the marginal column in the solution listing and set the marginal field in variables and equations to NA.

The outputs from these steps, including any possible error messages, are discussed in detail in chapter GAMS Output.

4.30.4 Programs with Several Solve Statements

Several solve statements can be processed in the same program. The next few subsections discuss various instances where several solve statements may be needed in the same file. If sequences of expensive or difficult models are to be solved, it might be useful to interrupt program execution and continue later. For details on this topic, see chapter The Save and Restart Feature.
4.30.4.1 Several Models

If there are different models then the solves may be sequential, as below. Each of the models in \([\text{PROLOG}]\) consists of a different set of equations, but the data are identical, so the three solves appear in sequence with no intervening assignments:

Solve nortonl using nlp maximizing z;
Solve nortonn using nlp maximizing z;
Solve nortone using nlp maximizing z;

When there is more than one solve statement in the program, GAMS uses as much information as possible from the previous solution to provide a starting point or basis in the search for the next solution.

4.30.4.2 Loop: One Model, Different Data

Multiple solves may also occur as a result of a solve statement within a \(\text{loop}\) statement. Loop statements are introduced and discussed in detail in chapter Programming Flow Control Features; here we show that they may contain a solve statement and thus lead to multiple solves within one model. The example from \([\text{MEANVAR}]\) computes the efficient frontier for return and variance for a portfolio selection problem at equidistance points.

\[
\begin{align*}
\text{loop(p(pp),} & \quad v.fx = v.min + (v.max-v.min)/(\text{card(pp)}+1)\text{*ord(pp);} \\
& \quad \text{Solve var1 maximizing m using nlp;} \\
& \quad xres(1,p) = x.l(i); \\
& \quad xres(\text{mean}',p) = m.l; \\
& \quad xres(\text{var}',p) = v.l; \\
\end{align*}
\]

The set \(p\) is a set of point between the minimum and maximum variance, it is the driving set of the loop. A variance variable \(v\) is fixed at a equidistance points. With each iteration through the loop another variance level is used, the \(\text{NLP}\) model var1 is solved for each iteration and the outputs are stored in the parameter \(xres(*,p)\), to be used later for reporting. As often for reporting purposes, the universal set \(*\) is used.

This example demonstrates how to solve the same model (in terms of variables and equations) multiple times with slightly different data. For such situations the Gather-Update-Solve-Scatter (GUSS) facility improves on the loop implementation by saving generation time and minimizing the communication with the solver. GUSS is activated by the additional keyword \text{scenario} in the solve statement followed by a set name that provides mapping information between parameters in the model and the scenario containers. A GUSS implementation of the loop would look as follows:

\[
\begin{align*}
\text{parameter vfx(p), px(p,i), pm(p);} \\
\text{set dict / p .scenario.''} \\
& \quad v \text{.fixed .vfx} \\
& \quad x \text{.level .px} \\
& \quad m \text{.level .pm /;} \\
& \quad vfx(p(pp)) = v.min + (v.max-v.min)/(\text{card(pp)}+1)\text{*ord(pp);} \\
& \quad \text{Solve var1 maximizing m using nlp scenario dict;} \\
& \quad xres(1,p) = px(p,i); \\
& \quad xres(\text{mean}',p) = pm(p); \\
& \quad xres(\text{var}',p) = vfx(p); \\
\end{align*}
\]
4.30.4.3 Customizing Solution Management: SolveOpt

It is important to consider how GAMS manages solutions if multiple models are solved. By default, GAMS merges subsequent solutions with prior solutions. This is not an issue if all models operate over the same set of variables. However, recursive procedures, different equation inclusions or logical conditions may cause only part of the variables or different variables to appear in the models to be solved. In such a case it might be useful to modify the solution management procedure using the model attribute or option SolveOpt.

4.30.4.4 Sensitivity or Scenario Analysis

Multiple solve statements can be used not only to solve different models, but also to conduct sensitivity tests, or to perform case (or scenario) analyses of models by changing data or bounds and then solving the same model again. While some commercial LP systems allow access to “sensitivity analysis” through GAMS it is possible to be far more general and not restrict the analysis to either solver or model type. This facility is even more useful for studying many scenarios since no commercial solver will provide this information.

An example of sensitivity testing is in the simple oil-refining model [MARCO]. Because of pollution control, one of the key parameters in oil refinery models is an upper bound on the sulfur content of the fuel oil produced by the refinery. In this example, the upper bound on the sulfur content of fuel oil was set to 3.5 percent in the original data for the problem. First, the model is solved with this value. Next, a slightly lower value of 3.4 percent is used and the model is solved again. Finally, the considerably higher value of 5 percent is used and the model is solved for the last time. Key solution values are saved for later reporting after each solve. This is necessary because a following solve replaces any existing values. The key solution values are the activity levels of the process level $z$, a variable that is defined over a set of processes $p$ and a set of crude oils $cr$. The complete sequence is:

```gams
parameter report(*,*,*) "process level report";

qs('upper','fuel-oil','sulfur') = 3.5 ;
Solve oil using lp maximizing phi;
report(cr,p,'base') = z.l(cr,p) ;
report('sulfur','limit','base') = qs('upper','fuel-oil','sulfur');

qs('upper','fuel-oil','sulfur') = 3.4 ;
Solve oil using lp maximizing phi ;
report(cr,p,'one') = z.l(cr,p) ;
report('sulfur','limit','one') = qs ('upper','fuel-oil','sulfur');

qs('upper','fuel-oil','sulfur') = 5.0 ;
Solve oil using lp maximizing phi ;
report(cr,p,'two') = z.l(cr,p) ;
report('sulfur','limit','two') = qs('upper','fuel-oil','sulfur');

Display report ;
```

Note that the parameter report is defined over the universal set or short universe. In general, the universe is useful when generating reports, otherwise it would be necessary to provide special sets containing the labels used in the report. Any mistakes made in spelling labels used only in the report should be immediately apparent, and their effects should be limited to the report. The parameter qs is used to set the upper bound on the sulfur content in the fuel-oil, and the value is retrieved for the report. Note that the display statement in the final line is introduced and discussed in detail in chapter The Display Statement. This example shows not only how simply sensitivity analysis can be done, but also how the associated multi-case reporting can be handled.

The output from the display statement is shown below. Observe that there is no production at all if the permissible sulfur content is lowered. The case attributes have been listed in the row SULFUR.LIMIT. Section Global Display Controls contains more details on how to arrange reports in a variety of ways.
### 4.30.4.5 Iterative Implementation of Non-Standard Algorithms

Another use of multiple solve statements is to permit iterative solution of different blocks of equations, most often using solution values from the first solve as data for the next solve. These decomposition methods are useful for certain classes of problems because the subproblems being solved are smaller, and therefore more tractable. One of the most common examples of such a method is the Dantzig-Wolfe decomposition.

An example of a problem that is solved in this way is a multi-commodity network flow problem in [DANWOLFE].

### 4.30.5 Choosing a Solver

After a model has been checked and prepared as described above, GAMS passes the model to a solver. When the GAMS system is installed default solvers for all model types are specified and these solvers are used if the user doesn’t specify anything else. It is easy to switch to other appropriate solvers provided the user has the corresponding license. There are multiple ways to switch solvers:

1. Using a command line parameter of the following form:
   
   ```
   gams mymodel model_type=solver name
   ```
   
   For example,
   
   ```
   gams mymodel lp=cbc
   ```

2. With an option command of the following form that is placed before the `solve` statement:
   
   ```
   Option model_type=solver_name;
   ```
   
   Here `option` is a keyword, `model_type` is the same model type that is used in the `solve` statement and `solver_name` is the name of one of the available solvers. For example,
   
   ```
   Option LP=cbc, NLP=conopt, MIP=cbc, MINLP=default;
   ```
   
   The `MINLP=default` switches back to the default solver for the MINLP model type.
3. Instead of providing a particular solver for a model type, the option `Solver` can be used to use a given solver for all model types this solver can handle.

    Option Solver=cbc;

4. (Re)running `gamsinst` at any time and altering the choice of default solver as described in the installation notes.

Note

A list of all solvers and current default solvers may be generated in the listing file with `Option SubSystems;`.

### 4.30.6 Making New Solvers Available with GAMS

This short section is to encourage those of you who have a favorite solver not available through GAMS. Linking a solver program with GAMS requires some programming skills and the use of libraries provided by GAMS. There is a collection of open source solver links to GAMS at the COIN-OR project [GAMSLinks](http://www.gams.com/links). The benefits of a link with GAMS to the developer of a solver are several. They include:

- Immediate access to a wide variety of test problems.
- An easy way of making performance comparisons between solvers.
- The guarantee that a user has not somehow provided an illegal input specification.
- Elaborate documentation, particularly of input formats, is not needed.
- Access to the existing community of GAMS users, for marketing or testing.

This completes the discussion of the model and solve statements.

### 4.31 Conditional Expressions, Assignments and Equations

#### 4.31.1 Introduction

This chapter deals with the way in which conditional assignments, expressions and equations are made in GAMS. The index operations already described are very powerful, but it is necessary to allow for exceptions of one sort or another. For example, heavy trucks may not be able to use a particular route because of a weak bridge, or some sectors in an economy may not produce exportable products. Exceptions such as these may easily be modeled with a logical condition combined with the dollar operator `$`, a very powerful feature of GAMS introduced in this chapter.

This chapter is organized as follows: We will introduce the general form of the dollar condition first and then we will focus on the various types of logical conditions. Next, we will discuss how dollar conditions are used to build conditional assignments, conditional indexed operations and conditional equations. We will conclude the chapter by showing that in certain cases conditions may be modeled using filtering sets instead of the dollar operator. Programming flow control features such as the if statement, the loop, the while statement, and the for statement are not covered in this chapter. These can be found in the chapter [Programming Flow Control Features](#).
4.31.2 The Dollar Condition

The dollar operator is one of the most powerful features in GAMS. The general syntax for a conditional expression is:

\[
term \; $ \; \text{logical\_condition}
\]

Here, \textit{term} can be a number, a (indexed) symbol, and also a complex expression. The dollar operator may be read as \textit{under the condition that} the following \textit{logical\_condition} evaluates to \textbf{TRUE} (or is unequal 0).

Consider the following simple condition, where \(a\) and \(b\) are scalars.

\[
\text{if (} b > 1.5 \text{), then } a = 2
\]

This can be written in GAMS using the dollar operator as follows.

\[
a \; $ \; (b > 1.5) \; = \; 2 \;
\]

Note that the \textit{term} is the scalar \(a\) and the \textit{logical\_condition} is the expression \((b > 1.5)\). If the condition is not satisfied, no assignment is made. To make it clear, this conditional assignment may be read as: \textit{‘given that \(b\) is greater than 1.5, \(a\) equals 2’}.

Logical conditions may take various forms, they are introduced in the next section. Conditional expressions may be used in the context of assignments, indexed operations and equations. These topics are covered in later sections of this chapter.

\textbf{Note}

Logical conditions used with the dollar operator cannot contain variables. However, \textit{variable attributes} are allowed.

4.31.3 Logical Conditions

Logical conditions are special expressions that evaluate to a value of either \textbf{TRUE} or \textbf{FALSE}. Logical conditions may be numerical expressions and numerical relations, they may refer to set membership and they may also contain acronyms. In the following subsections this is shown in the context of simple conditional assignments with the dollar operator on the left-hand side (compare section \textit{Dollar on the Left}).

In this section we use many examples to illustrate the concepts that are being introduced. In all these examples \(a\) and \(b\) are scalars, \(s, t, u\) and \(v\) are parameters, and \(i\) and \(j\) are sets.
4.31 Conditional Expressions, Assignments and Equations

4.31.3.1 Logical Conditions: Numerical Expressions

Numerical expressions may serve as logical conditions: a result of zero is treated as the logical value \texttt{FALSE} and a non-zero result is treated as the logical value \texttt{TRUE}. The following example illustrates this point.

\[
\begin{align*}
b &\implies (2\times a - 4) = 7; \\
\end{align*}
\]

Here the numerical expression \((2\times a - 4)\) is the logical condition. The numerical expression is zero if \(a\) equals 2, and non-zero otherwise. Hence the logical value of the expression is \texttt{FALSE} for \(a = 2\) and \texttt{TRUE} for all other values of \(a\). The assignment is only made if the numerical expression evaluates to \texttt{TRUE}, otherwise no assignment is made.

\textbf{Attention}

Values of the \textit{extended range arithmetic} such as \texttt{inf} are also allowed in logical conditions. If the result of a numerical expression used as a logical condition takes any of these values, the logical value is \texttt{TRUE}, even for e.g. \texttt{eps}, which is numerically 0.

Observe that \texttt{functions} are also allowed in logical conditions. If they evaluate to zero, the logical condition is \texttt{FALSE}, otherwise it is \texttt{TRUE}. Consider the following example:

\[
\begin{align*}
b &\implies \cos(a) = 7; \\
\end{align*}
\]

Note that the assignment is only made if the cosine of \(a\) does not equal zero.

4.31.3.2 Logical Conditions: Numerical Relational Operators

Numerical relational operators compare two numerical expressions and return a logical value. For completeness, all numerical relational operators are listed in \textit{Table 1}.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Operator</th>
<th>Alternative Notation</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strictly less than</td>
<td>(x &lt; y)</td>
<td>(x \lt y)</td>
<td>Returns \texttt{TRUE} if (x &lt; y), else returns \texttt{FALSE}.</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>(x \leq y)</td>
<td>(x \leq y)</td>
<td>Returns \texttt{TRUE} if (x \leq y), else returns \texttt{FALSE}.</td>
</tr>
<tr>
<td>Equal to</td>
<td>(x = y)</td>
<td>(x \ eq\ y)</td>
<td>Returns \texttt{TRUE} if (x = y), else returns \texttt{FALSE}.</td>
</tr>
<tr>
<td>Not equal to</td>
<td>(x \neq y)</td>
<td>(x \ ne\ y)</td>
<td>Returns \texttt{TRUE} if (x \neq y), else returns \texttt{FALSE}.</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>(x \geq y)</td>
<td>(x \ ge\ y)</td>
<td>Returns \texttt{TRUE} if (x \geq y), else returns \texttt{FALSE}.</td>
</tr>
<tr>
<td>Strictly greater than</td>
<td>(x &gt; y)</td>
<td>(x \ gt\ y)</td>
<td>Returns \texttt{TRUE} if (x &gt; y), else returns \texttt{FALSE}.</td>
</tr>
</tbody>
</table>

\textbf{Table 1:} Numerical Relational Operators

Consider the following examples.

\[
\begin{align*}
b &\implies (a < 0) = 10; \\
\end{align*}
\]
\[ b \$ (\text{sqr}(a) > a) = 12; \]
\[ a \$ (\text{sum}(i, s(i)) > 0 ) = 7; \]
\[ t(i) \$ (a <> 0) = t(i) + 1; \]

In the first line the logical condition is the relational expression \((a < 0)\). The assignment is only made if this expression is \text{TRUE}, that is, if the scalar \(a\) is negative. The logical condition in the second line is a bit more complex. It evaluates to \text{FALSE} if \(0 \leq a \leq 1\). For all other values of \(a\), it evaluates to \text{TRUE}. So the assignment is made for all values of \(a\), except for those values of \(a\) that are in the closed interval \([0, 1]\). Note that if \(a = -3\), then the logical condition in the first line will be \text{TRUE}, so \(b\) will become 10. In addition, the logical condition in the second line will be \text{TRUE}, so \(b\) will change to 12. The logical condition in the third line evaluates to \text{TRUE} if the sum of all values of the parameter \(a\) is strictly positive. Then \(a\) is assigned the value of 7. The assignment in the last line depends on whether \(a\) is non-zero. If \(a\) is zero no assignment is made, otherwise all entries of the parameter \(t\) are updated.

Note that acronyms may also be used with relational operators to build logical conditions. However, only the equality operator = and inequality operator <> are allowed in the context of acronyms, since they have no numerical values and the other operators would be meaningless. For an example, see section 

\text{Acronym Usage}.

### 4.31.3.3 Logical Conditions: Logical Operators

GAMS offers standard logical operators that may combine two or more logical conditions to build complex logical expressions. For example, if several expressions are required to be \text{TRUE} simultaneously, they may be connected with the operator \text{and}. The logical operators available in GAMS are listed in Table 2 and Table 3. Another way to construct complex logical conditions is by nesting them. For details, see subsection \text{Nested Dollar Conditions} below.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Alternative Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>not (x)</td>
<td></td>
<td>The logical condition (x) has to be \text{FALSE}, in order for the expression to be \text{TRUE}.</td>
</tr>
<tr>
<td>Logical conjunction</td>
<td>(x \text{ and } y)</td>
<td></td>
<td>Two logical conditions are \text{TRUE} simultaneously.</td>
</tr>
<tr>
<td>Logical disjunction</td>
<td>(x \text{ or } y)</td>
<td></td>
<td>At least one of two logical conditions applies.</td>
</tr>
<tr>
<td>Exclusive disjunction</td>
<td>(x \text{ xor } y)</td>
<td></td>
<td>Exactly one of two logical conditions applies.</td>
</tr>
<tr>
<td>Logical implication</td>
<td>(x \text{ imp } y)</td>
<td>(x \rightarrow y)</td>
<td>If the logical condition (x) is \text{TRUE} but at the same time the logical condition (y) is \text{FALSE}, then the whole expression is \text{FALSE}, in all other cases the expression evaluates to \text{TRUE}.</td>
</tr>
<tr>
<td>Logical equivalence</td>
<td>(x \text{ eqv } y)</td>
<td>(x \leftrightarrow y)</td>
<td>Both logical conditions are either \text{TRUE} simultaneously or \text{FALSE} simultaneously for the whole expression to be \text{TRUE}.</td>
</tr>
</tbody>
</table>

\text{Table 2: Logical Operators}

The logical values of these operators are summarized in the following truth table.

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>not (x)</th>
<th>(x \text{ and } y)</th>
<th>(x \text{ or } y)</th>
<th>(x \text{ xor } y)</th>
<th>(x \text{ imp } y)</th>
<th>(x \text{ eqv } y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{FALSE}</td>
<td>\text{FALSE}</td>
<td>\text{TRUE}</td>
<td>\text{FALSE}</td>
<td>\text{FALSE}</td>
<td>\text{FALSE}</td>
<td>\text{TRUE}</td>
<td>\text{TRUE}</td>
</tr>
</tbody>
</table>
Table 3: Truth Table of Logical Operators

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>not x</td>
<td>x and y</td>
<td>x or y</td>
<td>x xor y</td>
</tr>
<tr>
<td>FALSE</td>
<td></td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td></td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td></td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

The following somewhat artificial examples serve as illustration.

```plaintext
Set i / i1*i5 /;
Parameter s(i) / i1 3, i2 5, i4 8 /
  t(i) / i1*i4 13 /
  u(i) / i2 1 /
  v(i) / i1 7, i3 2 /

u(i) $ (not s(i)) = v(i);
u(i) $ (s(i) and u(i) and t(i)) = s(i);
u(i) $ (s(i) or v(i) or t(i)) = 4;
```

Note that there are three conditional assignments for the parameter u. In the first assignment the logical condition is (not s(i)). This condition holds for all entries of s that are not specified and therefore zero by default: s(i1) and s(i5). Hence u(i1) and u(i5) are assigned the values of v(i1) and v(i5) respectively. The value of v(i1) is 2 and the value of v(i5) is zero by default. After the first assignment we have u(i2)=1 and u(i3)=2, all other values of u are zero. Note that the logical condition failed for u(i2) and therefore its value remained unchanged. The logical condition in the second assignment is TRUE for those labels of the set i that have non-zero entries in the parameters s, u and t simultaneously. This condition holds only for i2. Therefore u(i2)=s(i2)=5 and all other values of u remain unchanged, resulting in non-zero values only for u(i2) and u(i3). The logical condition in the last assignment evaluates to TRUE for all labels of the set i that have at least one non-zero entry in the parameters s, v and t. This holds for all labels except for i5. Therefore u(i5) stays zero and all other values of u are changed to 4. These examples demonstrate the power of the dollar operator combined with logical operators. Even more complex logical conditions are possible; see subsection Mixed Logical Conditions below for details.

4.31.3.4 Logical Conditions: Set Membership and Set Functions

Apart from numerical and relational expressions, set membership and functions referencing set elements may be used as a logical condition. Consider the following example as illustration for set membership as logical condition.

```plaintext
Set i / i1*i5 /;
Parameter s(i) / i1 3, i2 5, i3 11, i4 8, i5 1 /
  t(i) / i1*i4 13 /
  u(i) / i2 1 /
  v(i) / i1 7, i3 2 /

u(i) $ (not s(i)) = v(i);
```

Note that the set j is a subset of the set i and that the parameter t is declared but not defined. The conditional expression t(i)$ j(i) in the last line restricts the assignment to the members of the subset j since only they satisfy the condition j(i). The values for t(i1) and t(i5) remain unchanged. In this case this means that they are zero (by default). Note that there is an alternative formulation for this type of conditional assignment; for details see subsection Filtering Sets in Assignments below.
Note

Only the membership of subsets and dynamic sets may be used as logical conditions.

The use of set membership as a logical condition is an extremely powerful feature of GAMS, see section Conditional Equations below for more examples.

Logical conditions may contain functions that return particular values depending on the position of elements in sets, the size of sets or the comparison of set elements to each other or text strings. These functions are presented in Table 4.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sameAs(element1,element2)</code> or <code>sameAs(&quot;text&quot;,element2)</code> or <code>sameAs(element1,&quot;text&quot;)</code></td>
<td>Returns TRUE if <code>element1</code> is identical to <code>element2</code> or <code>element1</code> is identical to <code>text</code>, and FALSE otherwise.</td>
</tr>
<tr>
<td><code>diag(element1,element2)</code> or <code>diag(&quot;text&quot;,element2)</code> or <code>diag(element1,&quot;text&quot;)</code></td>
<td>Same as function <code>sameAs</code>, but returns numerical value 1 for TRUE and 0 for FALSE.</td>
</tr>
<tr>
<td><code>card(set_name)</code></td>
<td>Returns the number of elements in a set. Note that this can also be called with a quoted text and returns the number of characters of that text in that case. For more details, see section The Card Operator.</td>
</tr>
<tr>
<td><code>ord(set_name)</code></td>
<td>Returns the relative position of an element in a set. Note that <code>ord</code> may be used only with one-dimensional, static, ordered sets. For more details, see section The Ord Operator.</td>
</tr>
</tbody>
</table>

Table 4: Functions Referencing Sets

In the following example we have two sets of cities and we want to know how many of them feature in both sets.

```gams
Set i / Beijing, Calcutta, Mumbai, Sydney, Johannesburg, Cairo /;
Set j / Rome, Paris, Boston, Cairo, Munich, Calcutta, Barcelona /;
Scalar b;
b = sum((i,j)$sameAs(i,j),1);
```

Note that in the assignment statement we `sum` over both sets and we use the function `sameAs` to restrict the domain of the indexed operation to those label combinations `(i,j)` where the function `sameAs` evaluates as TRUE. Thus only identical elements are counted.

Note that in this example the function `diag` could have been used with the same result.

The functions `ord` and `card` are frequently used to single out the first or last element of an ordered set. For example, we may want to fix a variable for the first and last elements of a set:

```gams
x.fx(i) $(ord(i) = 1) = 3;
x.fx(i) $(ord(i) = card(i)) = 7;
```

In the first assignment the variable `x` is fixed for the first element of the set `i` and in the second assignment `x` is fixed for the final element of `i`.


Note

As an alternative to the formulation above, one could also use the set attributes `first` and `last` to get the same result:

\[
x.fx(i) \ (i.first) = 3;
\]

\[
x.fx(i) \ (i.last) = 7;
\]

### 4.31.3.5 Numerical Values of Logical Conditions

We have seen that logical conditions may take the form of numerical expressions, expressions with relational operators, complex expressions using logical operators, set membership and set functions. However, GAMS does not have a Boolean data type.

Attention

GAMS follows the convention that the result of a relational operation is zero if the assertion is `FALSE`, and 1 if it is `TRUE`.

Consider the following example as illustration.

\[
x = (1 < 2) + (2 < 3);
\]

The expression to the right of the assignment evaluates to 2 since both logical conditions within parentheses are `TRUE` and therefore assume a value of 1. Note that this is different from the assignment below:

\[
x = (1 < 2) \ or \ (2 < 3)
\]

This assignment evaluates to 1, since both statements to the left and right of `or` are `TRUE` and therefore the whole expression is `TRUE`.

### 4.31.3.6 Mixed Logical Conditions

The building blocks introduced in the subsections above may be combined to generate more complex logical conditions. These may contain standard arithmetic operations, numerical relational operations and logical operations. All operations, their symbols and their order of precedence are given in Table 5. Note that 1 denotes the highest order of precedence and 7 denotes the lowest order of precedence. As usual, the default order of precedence holds only in the absence of parentheses and operators (symbols) on the same level are evaluated from left to right.

<table>
<thead>
<tr>
<th>Type of Operation</th>
<th>Operation</th>
<th>Operator</th>
<th>Order of precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard arithmetic operation</td>
<td>Exponentiation</td>
<td><code>**</code></td>
<td>1</td>
</tr>
<tr>
<td>Standard arithmetic operation</td>
<td>Multiplication, Division</td>
<td><code>*</code>, <code>/</code></td>
<td>2</td>
</tr>
<tr>
<td>Standard arithmetic operation</td>
<td>Unary operators: Plus, Minus</td>
<td><code>+</code>, <code>-</code></td>
<td>3</td>
</tr>
<tr>
<td>Standard arithmetic operation</td>
<td>Binary operators: Addition, Subtraction</td>
<td><code>+</code>, <code>-</code></td>
<td>3</td>
</tr>
<tr>
<td>Numerical Relational operation</td>
<td>All</td>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>=</code>, <code>&lt;&gt;</code>, <code>&gt;=</code>, <code>&gt;</code>, <code>&lt;</code></td>
<td>4</td>
</tr>
<tr>
<td>Logical operation</td>
<td>Negation</td>
<td><code>not</code></td>
<td>5</td>
</tr>
<tr>
<td>Logical operation</td>
<td>Logical Conjunction</td>
<td><code>and</code></td>
<td>6</td>
</tr>
<tr>
<td>Logical operation</td>
<td>All other logical operators</td>
<td><code>or</code>, <code>xor</code>, <code>imp</code>, <code>eqv</code></td>
<td>7</td>
</tr>
</tbody>
</table>
Table 5: Complete Hierarchy for Operator Precedence in GAMS

Note

We recommend to use parentheses rather than relying on the order of precedence of operators. Parentheses prevent errors and make the intention clear.

Consider the following example:

\[x - 5*y \text{ and } z - 5\]

\[(x - (5*y)) \text{ and } (z-5)\]

These two complex logical conditions are equivalent. However, the parentheses make the second expression easier to understand.

Some simple examples of complex logical conditions, their numerical values and their logical values are given in Table 6.

<table>
<thead>
<tr>
<th>Logical Condition</th>
<th>Numerical Value</th>
<th>Logical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1 &lt; 2) + (3 &lt; 4))</td>
<td>2</td>
<td>TRUE</td>
</tr>
<tr>
<td>((2 &lt; 1) \text{ and } (3 &lt; 4))</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>((4+5 - 3) + (10/8))</td>
<td>18.25</td>
<td>TRUE</td>
</tr>
<tr>
<td>((4+5 - 3) \text{ or } (10 - 8))</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>((4 \text{ and } 5) + (2+3 &lt;= 6))</td>
<td>2</td>
<td>TRUE</td>
</tr>
<tr>
<td>((4 \text{ and } 0) + (2+3 &lt; 6))</td>
<td>0</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Table 6: Examples of Complex Logical Conditions

4.31.3.7 Nested Dollar Conditions

An alternative way to model complex logical conditions is by nesting them. The syntax is:

\[\text{term } \$(\text{logical\_condition1\$\text{(logical\_condition2\$\ldots)})}\]

Note that in nested dollar conditions all succeeding expressions after the dollar operator must be enclosed in parentheses. The nested expression is equivalent to the following conditional expression that uses the logical operator \textbf{and} instead of the nesting:

\[\text{term } \$(\text{logical\_condition1 \text{ and logical\_condition2 \text{ and } \ldots})}\]

Consider the following example. Note that \(i, j(i)\) and \(k(i)\) are sets, and \(u(i)\) and \(v(i)\) are parameters.

\[u(i) \$(j(i) k(i)) = v(i) \]

The assignment will be made only for those members of the set \(i\) that are members of both sets \(j\) and \(k\). Note the position of the parentheses in the dollar condition. The assignment statement above can be rewritten in the following way.

\[u(i) \$(j(i) \text{ and } k(i)) = v(i) \]

Note

We recommend to use the logical \textbf{and} operator instead of nesting dollar operators, because this formulation is easier to read.
4.31 Conditional Expressions, Assignments and Equations

4.31.4 Conditional Assignments

A conditional assignment is an assignment statement with a dollar condition on the left-hand side or on the right-hand side. Most examples until now were conditional assignments with the dollar operator on the left.

Attention

The effect of the dollar condition is significantly different depending on which side of the assignment it is located.

The next two subsections describe the use of the dollar condition on each side of the assignment. Note that in many cases it may be possible to use either of the two forms of the dollar condition to describe an assignment. We recommend to choose the clearer formulation.

Note that if the logical condition in an assignment statement refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section Filtering Sets in Assignments below.

4.31.4.1 Dollar on the Left

If the dollar condition is on the left-hand side of an assignment, an assignment is made only in case the logical condition is satisfied. If the logical condition is not satisfied then no assignment is made and the previous content of the parameter on the left will remain unchanged. In case the parameter on the left-hand side of the assignment has not previously been initialized or assigned any values, zeros will be used for any label for which the assignment was suppressed.

Consider the following example adapted from [CHENERY]. Note that the parameter $sig$ has been previously defined in the model.

\[
\text{rho}(i) \;\$\; (\text{sig}(i) \not= 0) = (1./\text{sig}(i)) - 1. ;
\]

In this assignment $\text{rho}(i)$ is calculated and the dollar condition on the left protects against dividing by zero. If any of the values associated with the parameter $\text{sig}$ turns out to be zero, no assignment is made and the previous values of $\text{rho}(i)$ remain. As it happens, $\text{rho}(i)$ was not previously initialized, and therefore all the labels for which $\text{sig}(i)$ is zero will result in a value of zero.

Now recall the convention that non-zero implies TRUE and zero implies FALSE. The assignment above could therefore be written as:

\[
\text{rho}(i) \;\$\; \text{sig}(i) = (1./\text{sig}(i)) - 1. ;
\]

In the following examples $i$ is a set and $s$ and $t$ are parameters.

\[
\text{s}(i) \;\$\; \text{t}(i) = \text{t}(i);
\]

\[
\text{s}(i) \;\$\; ((\text{t}(i)-1) > 0) = \text{t}(i)**0.5;
\]

Note that the first assignment is suppressed if the value of the parameter $t$ equals zero. The second assignment is suppressed for values of the parameter $t$ that are smaller or equal to 1.

Note

The first of the last two examples is special in the way that the logical condition used with the dollar on the left and the expression on the right hand side of the assignment are actually the same. This allows to use a shorter notation using the combined $\$=$ assignment. That assignment is equivalent to the following one (which called sparse assignment):

\[
\text{s}(i) \;\$\;= \text{t}(i);
\]
4.31.4.2 Sparse Assignments

Sparse assignments will assign a value to the left-hand side of the = sign only if the right-hand side is nonzero. This behavior is triggered using the $=$ notation as in the following example.

```
Set i /a,b,c/
Parameter d1(i) "Initial data" /a 1, b 1, c 1/
   d2(i) "Data to be used to overwrite d1" /a 0, b -2 /
   d3(i) "Empty data parameter";

* Initialize d3 to d1
   d3(i) = d1(i);
* Use dollar on the left to overwrite d3 with non-zero values from d2
   d3(i)$d2(i) = d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;

* Re-Initialize d3 to d1
   d3(i) = d1(i);
* Use sparse assignment to overwrite d3 with non-zero values from d2
   d3(i) $= d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;
```

Note that in the final assignments above, the values of the parameter d3(i) are replaced with values from parameter d2(i) only if the entry in d2(i) is non-zero, the other values are left untouched. Both assignments are equivalent.

4.31.4.3 Dollar on the Right

If the dollar condition is on the right-hand side of an assignment statement, an assignment will always be made. In case the logical condition is not satisfied the value of zero is assigned.

Above we had the following simple conditional assignment:

```
a $ (b > 1.5) = 2;
```

Now we move the dollar condition to the right-hand side:

```
a = 2 $ (b > 1.5) ;
```

This is equivalent to:

```
if (b > 1.5) then (a = 2), else (a = 0)
```

Note that an if-then-else type of construct is implied, but the else operation is predefined and never made explicit. The else could be made explicit with the following formulation:

```
a = 2 $ (b > 1.5) + 0 $ (b <= 1.5) ;
```
The use of this feature is more apparent in instances when an else condition needs to be made explicit. Consider the next example adapted from the fertilizer model [FERTD]. The set $i$ is the set of plants, and we are calculating $mur(i)$, the cost of transporting imported raw materials. In some cases a barge trip must be followed by a road trip because the plant is not alongside the river and we must combine the separate costs. The assignment is:

\[
mur(i) = (1.0 + 0.0030 \cdot ied(i,'barge')) \cdot ied(i,'barge') + (0.5 + 0.0144 \cdot ied(i,'road')) \cdot ied(i,'road');
\]

This means that if the entry in the distance table $ied$ is not zero, then the cost of shipping using that link is added to the total cost. If there is no distance entry, there is no contribution to the cost, presumably because that mode is not used.

Consider another example for a conditional assignment with the dollar operator on the right:

\[b = \text{sum}(i,t(i)) \cdot (a > 0) + 4;\]

Here $a$ and $b$ are scalars, $i$ is a set and $t$ is a parameter. If the scalar $a$ is positive, the scalar $b$ is assigned the sum of all values of the parameter $t$ plus 4. If $a$ is zero or negative, $b$ becomes just 4. Note that the sum is only computed if the condition holds, this potentially makes the program faster.

### 4.31.5 Conditional Indexed Operations

We have seen how exceptions in assignments are modeled with dollar conditions. Dollar conditions are also used in indexed operations, where they control the domain of operation. This is conceptually similar to the conditional assignment with the dollar on the left.

Consider the following example adapted from [GTM], a gas trade model for interrelated gas markets. Here the set $i$ contains supply regions and the parameter $supc$ models supply capacities. The scalar $tsupc$ is computed with the following statement:

\[tsupc = \text{sum}(i \cdot (supc(i) \neq \text{inf}), supc(i));\]

This assignment restricts the sum to the finite values of the parameter $supc$.

In indexed operations the logical condition is often a set. This set is called the conditional set and assignments are made only for labels that are elements of the conditional set. This concept plays an important role in dynamic sets.

Multi-dimensional sets are introduced in section Multi-Dimensional Sets. In the example used there a two-dimensional set is used to define the mapping between countries and ports. Another typical example for a multi-dimensional set is a set-to-set mapping that defines the relationship between states and regions. This is useful for aggregating data from the state to the regional level. Consider the following example:

Sets $r$ "regions" / north, south /
$s$ "states" / florida, texas, vermont, maine /
corr(r,s) / north.(vermont,maine) /
  south.(florida,texas) /;

Parameter $y(r)$ "income for each region"
income(s) "income of each state"
  / florida 4.5, vermont 4.2, texas 6.4, maine 4.1 /;
The set \texttt{corr} links the states to their respective regions, the parameter \texttt{income} is the income of each state. The parameter \texttt{y} is computed with the following assignment statement:

\[
y(r) = \text{sum}(s \in \texttt{corr}(r,s), \texttt{income}(s)) ;
\]

The conditional set \texttt{corr}(r,s) restricts the domain of the summation: for each region \texttt{r} the summation over the set \texttt{s} is restricted to the label combinations \((r,s)\) that are elements of the set \texttt{corr}(r,s). Conceptually, this is analogous to the Boolean value \texttt{TRUE} or the arithmetic value \texttt{non-zero}. The effect is that only the contributions of \texttt{vermont} and \texttt{maine} are included in the total for \texttt{north}, and \texttt{south} is the sum of the incomes from only \texttt{texas} and \texttt{florida}.

Note that the summation above can also be written as:

\[
y(r) = \text{sum}(s, \texttt{income}(s) \in \texttt{corr}(r,s));
\]

In this formulation the parameter \texttt{income} is controlled by the conditional set \texttt{corr} instead of the index \texttt{s}. Note that both formulations yield the same result, but the second alternative is more difficult to read.

In the next example the logical condition on the right-hand side appears in the context of an indexed operation, but it does not refer to the index. Thus it is treated similarly to a dollar on the right logical condition. Note that \texttt{a} and \texttt{b} are scalars, \texttt{i} is a set and \texttt{s} is a parameter.

\[
b = \text{sum}(i \in a, s(i));
\]

Here the scalar \texttt{b} is assigned the sum of the values of the parameter \texttt{s} only if \texttt{a} does not equal zero. Otherwise \texttt{b} will be assigned the value zero. Observe that the following alternative formulation will generally be faster than the assignment above:

\[
b = \text{sum}(i, s(i)) \in a;
\]

Note that if the logical condition in the context of indexed operations refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section \texttt{Filtering Controlling Indices in Indexed Operations} below.

\subsection*{4.31.6 Conditional Equations}

The dollar operator is also used for exception handling in equations. The next two subsections discuss the two main uses of dollar operators in the context of equations: in the body of an equation and over the domain of definition.
4.31.6.1 Dollar Operators within the Algebra of Equations

A dollar operator in the algebraic formulation of an equation is analogous to the dollar control on the right of assignments, as presented in section Dollar on the Right. Assuming that "the right" means the right of the '...' then the analogy is even closer. As in the context of assignments, an if-else operation is implied. It is used to exclude parts of the definition from some of the generated constraints.

Consider the following example adapted from the production model [CHENERY].

Set i "sectors" / light-ind, food+agr, heavy-ind, services /
    t(i) "tradables" / light-ind, food+agr, heavy-ind /;
Variables x(i) "quantity of output"
    y(i) "final consumption"
    e(i) "quantity of exports"
    m(i) "quantity of imports";
Equations mb(i) "material balance";

mb(i).. x(i) =g= y(i) + (e(i) - m(i)) $ t(i) ;

Note that in the equation definition in the last line, the term \((e(i) - m(i))\) on the right-hand side of the equation is added only for those elements of the set \(i\) that also belong to the subset \(t(i)\), so that the element services is excluded.

Further, conditional indexed operations may also feature in expressions in equation definitions. The supply balance equation from the gas trade model [GTM] is an example. Note that the set \(i\) contains the supply regions, the set \(j\) contains the demand regions, and the two-dimensional set \(ij\) is the set of feasible links; the variable \(x\) denotes the shipment of natural gas and the variable \(s\) denotes the regional supply.

\[
sb(i).. \sum(j \; ij(i,j), \; x(i,j)) =l= \; s(i) ;
\]

Similar to the assignment example seen before, the conditional set \(ij(i,j)\) restricts the domain of the summation: for each supply region \(i\) the summation over the demand regions \(j\) is restricted to the label combinations \((i,j)\) that are elements of the set of feasible links \(ij(i,j)\).

4.31.6.2 Dollar Control over the Domain of Definition

In case constraints should only be included in the model if particular conditions are met, a dollar condition in the domain of definition of an equation may be used to model this restriction. Such a dollar condition is analogous to the dollar control on the left of assignments. Assuming that "the left" means the left of the symbols '...' then the analogy is even closer.

Note

The dollar control over the domain of definition of equations restricts the number of constraints generated to less than the number implied by the domain of the defining sets.

Consider the following example adapted from the Andean fertilizer model [ANDEAN] :

\[
gple(w,wp,te) \; gple(w,wp).. \; yw(w,te) - yw(wp,te) =l= \; dpack;
\]
Here \( u, \wp \) and \( te \) are sets, \( \text{ple} \) is a two-dimensional parameter, \( \yw \) is a variable and \( \text{dpack} \) is a scalar. Note that the dollar condition restricts the first two indices of the domain of the equation to those label combinations that have non-zero entries in the two-dimensional parameter \( \text{ple} \).

Sometimes the desired restriction of an equation may be achieved either way: through a condition in the algebra or a condition in the domain of definition. Compare the following two lines, where \( \text{eq1} \) and \( \text{eq2} \) are equations, \( i \) and \( j \) are sets, \( b \) is a scalar, \( s \) is a parameter and \( x \) is a two-dimensional variable.

\[
\text{eq1}(i) \quad b.. \quad \sum(j, x(i,j)) =g= -s(i); \\
\text{eq2}(i) \quad \sum(j, x(i,j)) \quad b =g= -s(i) \quad b;
\]

In the first line the dollar condition is in the domain of definition, in the second line the dollar conditions are in the algebraic formulation of the equation. If \( b \) is non-zero, the generated equations \( \text{eq1} \) and \( \text{eq2} \) will be identical. However, if \( b \) is 0, no equation \( \text{eq1} \) will be generated, but for each \( i \) we will see a trivial equation \( \text{eq2} \) of the form \( 0 =g= 0; \).

Note that if the logical condition in the domain of definition of an equation refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section Filtering the Domain of Definition below.

### 4.31.7 Filtering Sets

If the logical condition refers to set membership, the restriction modeled with a dollar condition may sometimes be achieved without the dollar operator. Consider the following statement, where \( i \) and \( j(i) \) are sets, and \( u \) and \( s \) are parameters:

\[
u(i) \quad j(i) = s(i) ;
\]

Note that the assignment is made only for those elements of the set \( i \) that are also elements of the subset \( j \). This conditional assignment may be rewritten in a shorter way:

\[
u(j) = s(j) ;
\]

In this statement the assignment has been filtered through the condition without the dollar operator by using the subset \( j \) as the domain for the parameters \( u \) and \( s \). This formulation is cleaner and easier to understand. It is particularly useful in the context of multi-dimensional sets (tuples), and it may be used in assignments, indexed operations and the domain of definition of equations.

### 4.31.7.1 Filtering Sets in Assignments

Suppose we want to compute the transportation cost between local collection sites and regional transportation hubs for a fictional parcel delivery service. We define sets for the collection sites and transportation hubs and a two-dimensional set where the collection sites are matched with their respective hubs:
Set i "local collection sites" / miami, boston, chicago, houston, phoenix /
   j "regional transportation hubs" / newyork, detroit, losangeles, atlanta /;

Set r(i,j) "regional transportation hub for each local collection site" /
   boston .newyork
   miami .atlanta
   houston .atlanta
   chicago .detroit
   phoenix .losangeles /;

Table distance(i,j) "distance in miles"
   newyork detroit losangeles atlanta
   miami 1327 1387 2737 665
   boston 216 699 3052 1068
   chicago 843 275 2095 695
   houston 1636 1337 1553 814
   phoenix 2459 1977 398 1810;

Parameter factor "cost estimate per unit mile"
   shipcost(i,j) "cost of transporting parcels from a local collection site to a regional hub"

factor = 0.009 ;
shipcost(i,j) $ r(i,j) = factor*distance(i,j) ;

The distance between collection sites and transportation hubs is given in the table. The last line is a
conditional assignment for the parameter shipcost. This assignment is only made if the label combination
(i,j) is an element of the set r. Note that in each instance the indices i and j appear together. Thus
the assignment may be simply written as:

shipcost(r) = factor*distance(r) ;

Note that the assignment is explicitly restricted to the members of the set r; the dollar operator is not
necessary. Observe that if the indices i or j appear separately in any assignment, the above simplification
cannot be made. For example, consider the case where the shipping cost depends not only on the parameter
factor and the distance between collection sites and regional hubs, but also on the congestion at the
regional hub. We introduce a new parameter congestfac that models the congestion at each regional
hub and is indexed only over the set j:

Parameter congestfac(j) "congestion factor" /
   newyork 1.5
   detroit 0.7
   losangeles 1.2
   atlanta 0.9/ ;

The new cost of shipment is computed as follows:

shipcost(i,j) $ r(i,j) = factor*congestfac(j)*distance(i,j) ;

Note that this conditional assignment cannot be reformulated as:

shipcost(r) = factor*congestfac(j)*distance(r) ;

In the representation above the index j appears on the right-hand side, but not on the left-hand side.
GAMS will flag this assignment as an error. However, the following representation will work:

shipcost(r(i,j)) = factor*congestfac(j)*distance(r) ;

In this formulation the set r is explicitly denoted as a tuple of the sets i and j. The set j may then
appear on the right-hand side.
4.31.7.2 Filtering Controlling Indices in Indexed Operations

Similarly, the controlling indices in indexed operations may be filtered through the conditional set without the use of the dollar operator. We continue with the shipping cost example from the last subsection. The total cost of shipment is obtained through the equation that follows. We also include the variable definitions for clarity.

Variable shipped(i,j), totcost ;
Equation costequ ;

costequ.. totcost =e= sum((i,j) $ r(i,j), shipcost(i,j)*shipped(i,j));

Here the variable shipped is the number of parcels shipped from the local collection site i to the regional transportation hub j, and the variable totcost is the total cost of all shipments. Note that the summation in the equation is restricted to the label combinations that are elements of the set r. Alternatively, the equation above may be written as:

costequ.. totcost =e= sum(r, shipcost(r)*shipped(r));

In this formulation the summation is performed explicitly only over the elements of the set r, no dollar condition is necessary. However, if the expression in the equation included a term dependent only on index j, then we would have to reformulate differently. Suppose the equation included also the congestion factor congestfac that is indexed only over j:

cost.. totcost =e= sum((i,j) $ r(i,j), factor*congestfac(j)*distance(i,j)*shipped(i,j));

In this case the equation needs to be simplified in the following way:

cost.. totcost =e= sum(r(i,j), factor*congestfac(j)*distance(r)*shipped(r));

Like before, the domain of the indexed operation sum is the set r. But this time the domain of r has to be named explicitly, so that the parameter congestfac which is indexed only over the set j is permitted in the scope of the indexed operation. Note that this reasoning is analogous to the reasoning for filtering sets in assignments in the subsection above.

4.31.7.3 Filtering the Domain of Definition

The rules for filtering sets that we have introduced in subsections Filtering Sets in Assignments and Filtering Controlling Indices in Indexed Operations also apply in the context of equation domains. We continue with the parcel transport example introduced above and add a binary variable bin, the parameter bigM and the equation connect to the model. Recall that shipped(i,j) is a variable and r(i,j) is a set.

Parameter bigM(i,j) ;
Binary variable bin(i,j) ;
Equation connect(i,j) ;

connect(i,j) $ r(i,j).. shipped(i,j) =l= bigM(i,j)*bin(i,j) ;
The dollar condition restricts the domain of definition of the equation \texttt{connect} to those label combinations of the sets \texttt{i} and \texttt{j} that are elements of the set \texttt{r}. The equation relates the continuous variable \texttt{shipped}(i,j) to the binary variable \texttt{bin}(i,j). Note that each domain in the equation is the index pair \texttt{(i,j)}. So the equation may be simplified as follows:

\begin{verbatim}
connect(r).. shipped(r) =l= bigM(r)*bin(r) ;
\end{verbatim}

In this formulation the domain of the equation is explicitly restricted to the members of the set \texttt{r}, without the use of a dollar condition. Note that if the right-hand side of the equation contained any term that was indexed over \texttt{i} or \texttt{j} separately, then the domain of definition of the equation would have to be simplified as:

\begin{verbatim}
connect(r(i,j))
\end{verbatim}

The reasoning is the same as in the case of assignments and indexed operations.

### 4.32 The Display Statement

#### 4.32.1 Introduction

The \texttt{display} statement in GAMS is a quick way to write data into the listing file. It does not provide a publication quality reporting function, but is instead aimed at functionality that is easy to use and provides graceful defaults. These defaults may be modified by the user to control the layout and appearance of the output. Both defaults and customization options are presented in this chapter. For information on the listing file, see chapter \texttt{GAMS Output}. In addition to the \texttt{display} statement, GAMS offers the \texttt{put writing facility}, a more sophisticated way to generate reports, where the user has much more flexibility and control over the output of individual items.

#### 4.32.2 The Syntax

In general, the syntax for the \texttt{display} statement in GAMS is as follows:

\begin{verbatim}
display ident | quoted text {, ident | quoted text};
\end{verbatim}

The keyword \texttt{display} indicates that this is a \texttt{display} statement. It is followed by an \texttt{identifier}. If the identifier is a set or a parameter, only the name of the set or parameter itself is specified. If the identifier is a variable, an equation or a model, it must be followed by a suffix, i.e. a \texttt{variable attribute}, \texttt{equation attribute} or \texttt{model attribute} respectively. As usual, \texttt{quoted text} must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines. For conditional \texttt{displays} a dollar condition may follow the keyword \texttt{display}.

The output produced by a display statement consists of labels and data. For multi-dimensional sets the keyword \texttt{yes} (indicating membership) is used instead of values.
Note

Only the non-default values are displayed for all data types.

The default value is generally zero. Exceptions are some upper and lower bounds of variables and
equations (the attributes .lo and .up). Their default values are listed in sections Variable Attributes and
Equation Attributes respectively.

The syntax of the display statement is illustrated by the following example:

```plaintext
Sets  s   / s1*s4 /  
t   / t5*t7 /  
    st(s,t) / s1.t5, s1.t7, s2.t6, s3.t7, s4.t5 / ;
Parameters p(s)  / s1 0.33, s3 0.67 /  
    q(t)  / t5 0.33, t7 0.67 / ;
Variable v(s,t) ;  
v.l(s,t) = p(s)*q(t);  
display 'first a simple set', s, 'then a two-dimensional set', st, 'then a parameter', p,  
'then the activity level of a variable', v.l;
```

The display statement will generate the following lines in the listing file:

```plaintext
----- 8 first a simple set
----- 8 SET s
     s1, s2, s3, s4

----- 8 then a two-dimensional set
----- 8 SET st
     t5    t6    t7
     s1 YES  YES
     s2 YES
     s3 YES
     s4 YES

----- 8 then a parameter
----- 8 PARAMETER p
     s1 0.330, s3 0.670

----- 8 then the activity level of a variable
----- 8 VARIABLE v.L
     t5     t7
     s1 0.109 0.221
     s3 0.221 0.449
```
In the case of multi-dimensional identifiers like the set \texttt{st} and the level values of the variable \texttt{v}, the data is by default reported in a tabular form that is easy to read. Note that only the non-default values are displayed. To display zero values, the special value \texttt{EPS} can be used. In the example above, all values of the parameter \texttt{p} can be displayed by adding the following lines.

```gams
display 'display all values';
p(s)$(not p(s)) = EPS;
display p;
```

The following lines will be displayed in the listing file.

```gams
11 display all values
13 PARAMETER p
s1 0.330, s2 EPS, s3 0.670, s4 EPS
```

An alternative to displaying \texttt{EPS} is to use \texttt{acronym} but computations cannot be done with the resulting parameters.

### 4.32.3 Displaying Multi-Dimensional Identifiers: Label Order

We have seen that two-dimensional identifiers are displayed in table format. The default layout for displaying multi-dimensional identifiers is summarized in Table 1. Note that the default format may be modified with the local display control. The figures in the table refer to the index position in the domain list of the identifier. For example, if we display \texttt{x}, where \texttt{x} has been declared as \texttt{x(i,j,k,l)}, then the \texttt{i} labels (the first index) will by default be associated with the individual subtables, the \texttt{j} and \texttt{k} with the row labels and the \texttt{l} (the fourth and last index) with the column headings.

<table>
<thead>
<tr>
<th>Number of Indices</th>
<th>Subtable</th>
<th>Index Position(s) on the Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>list format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>1,2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2,3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1,2</td>
<td>3,4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1,2,3</td>
<td>4,5</td>
<td>6</td>
</tr>
</tbody>
</table>

**Table 1:** Default Layout for Displaying Multi-Dimensional Identifiers

For 7 to the maximum number of indices the natural progression is followed. For the current maximum number of indices, see Dimensions. The labels vary slowest for the first index position and quickest for the highest. Within each index position the order of the label is the GAMS entry order of the labels. Some users may need to manipulate the order of the labels.

Note

The only way to change the order in which the labels for each index position appear in the display output is to change the order of appearance of the labels in the GAMS program.
This is most easily done by declaring a set whose only purpose is to list all the labels in the order that is needed. Make this set the very first declaration in the GAMS program.

Note that the order of the indices in an identifier is always as the order in the declaration statement. One can declare them in the order that is found appealing, or make an assignment to a new identifier with a different order, as illustrated in the example below.

### 4.32.3.1 Example for Multi-Dimensional Displays

Consider the following example. The parameter \( x \) has four dimensions or index positions. It is initialized using parameter format and then displayed as shown below.

Set \( i \) first index / first, second /
   \( j \) second index / one, two, three /
   \( k \) third index / a, b /
   \( l \) fourth index / i, ii / ;

Parameter \( x(i,j,k,l) \) a four dimensional structure /
second.one.a.i 0.00013, first .three.b.i -6.3161
first .one.b.i 5.63559, second.two .b.i 20000.00
second.one.b.ii -17.29948, first .two .b.ii 10.3457
first .two.a.ii 0.02873, second.one .a.ii 1.0037
second.two.a.ii +inf, first .two .a.i -2.9393
first .one.a.ii 0.00000 /

\( y(k,l,i,j) \) a four dimensional structure with changed index order;
\( y(k,l,i,j) = x(i,j,k,l) \);
display \( x, y \);

For variable \( x \) the code fragment produces the following output:

```
---- 16 PARAMETER x a four dimensional structure

INDEX 1 = first

   i     ii

   one .b  5.636
   two .a -2.939  0.029
   two .b  10.346
   three.b -6.316

INDEX 1 = second

   i     ii

   one.a 1.300000E-4  1.004
   one.b  -17.299
   two.a  +INF
   two.b  20000.000
```
Notice that there are two subtables, one for each label in the first index position. Note further that the order of the labels is not the same as in the input data list of the parameter x, but is according to the label order in the driving sets. Observe that the numbers closer in magnitude to one are displayed in fixed format, rounded to three digits (the default). This default may be modified with display controls, the topic of the next section. In contrast, the especially small value appears in E-format: it would display as all zeros in fixed format. Note also that the zero in the list for \( x('first', 'one', 'a', 'ii') \) has vanished, since default values are not shown. Similarly, rows and columns containing only zero values are suppressed in each subtable separately.

Notice furthermore how parameter y can be used to display the same data with a different index order.

### 4.32.4 Display Controls

GAMS allows the user to modify the number of row and column labels in the display listing as well as the accuracy of the data being displayed. There are global and local display controls. Global display controls allow the user to affect more than one display statement. The local display control may be used to override the global controls if specific data need to be listed in a particular format. Both controls are expressed with option statements.

#### 4.32.4.1 Global Display Controls

The first option for global display controls specifies the number of digits shown after the decimal point. It affects the numbers appearing in all display output following the option statement, unless it is changed for a specific identifier as shown below. The general form of this option statement is:

\[
\text{option decimals = value;}
\]

The keyword `option` indicates that this is an option statement, `decimals` is the specific option and `value` is an integer between 0 and 8. The value 0 supresses decimals as well as the decimal point. Note that the default is `decimals=3`, and that setting this option does not change the field width, just the number of decimals. Increasing `decimals` results in more digits being displayed; this may in turn cause numbers to appear in E-format if the longer fixed-format result would exceed the field width.

A word about rounding for display purposes is in order. In the event of ties, the display routines use the round-to-even tie-breaking rule. So for example, with `decimals=0`, both 1.5 and 2.5 will display as 2. Similarly, with `decimals=1`, 2.25 displays as 2.2 while 2.75 displays as 2.8.

Consider the following extension to the example discussed in the previous section.

\[
\text{option decimals = 1; display x;}
\]

GAMS will display numbers using E-format where necessary to avoid exceeding the display width or showing all zeros. The output is as follows:

\[
---
14 \text{ PARAMETER} \ x \ \text{a four dimensional structure}
\]

\[
\text{INDEX} \ i = \text{first}
\]

\[
\begin{array}{ll}
    \text{one} & \text{.b} & 5.6
\end{array}
\]
Note that GAMS overrode the decimal choice for $x('first', 'two', 'a', 'ii$) to ensure that small numbers are not displayed as zeroes.

Another option for global display control modifies the width of the display. This is particularly important for longer label names. Label names that are headers of columns are cut off after 9 characters when data is displayed in table format. The following option may be used to display longer names in full:

```
option dispwidth = value;
```

The name of the option is `dispwidth` and `value` is a positive integer smaller or equal to 31. An illustrative example follows.

```
Set i 'South African cities' / Johannesburg, Cape_Town /
    j 'other African cities' / Maputo, Dar_es_Salaam /
    f(i,j) 'direct flight connections' / Johannesburg.Maputo, Johannesburg.Dar_es_Salaam, Cape_Town.Maputo;

display f;
```

The display statement generates the following output:

```
---- 4 SET f direct flight connections
       Maputo Dar_es_Salaam
Johannesburg         YES         YES
Cape_Town            YES
```

Note that only the label 'Dar_es_Salaam' is cut off since it is heading a column. The label 'Johannesburg' is not cut off as it is the label for a row. Now we add the following line to change the width of the display statement:

```
option dispwidth = 15; display f;
```

As expected, the output changes and the label 'Dar_es_Salaam' is no longer cut off:

```
---- 5 SET f direct flight connections
       Maputo     Dar_es_Salaam
Johannesburg         YES         YES
Cape_Town            YES
```
4.32.4.2 Local Display Control

It is often useful to control the number of decimals for specific identifiers separately. The following option statement accomplishes this:

option ident:d-value;

Ident denotes the name of the identifier and d-value is an integer in the range 0 to 8. The number of decimal places shown on all displays of ident that follow will be d-value.

This option statement can be extended to control the layout of the data in addition to the decimal places. The general form is as follows:

option ident:d-value:r-value:c-value ;

Here r-value denotes the number of index positions that are combined to form the row label and c-value means the number of indices on the column headers.

The example discussed in the previous section is further extended in order to illustrate the local display control.

option x:5:3:1; display x;

These two statements generate the following output:

```
---- 17 PARAMETER x a four dimensional structure
       i    ii
  first .one .b   5.63559
  first .two .a  -2.93930  0.02873
  first .two .b    10.34570
 first .three .b -6.31610
second .one .a   0.00013   1.00370
second .one .b    -17.29948
second .two .a     +INF
second .two .b 20000.00000
```

Note that five decimal places are shown, three labels are used to mark the rows and one label acts as head of the column. Since this is a four-dimensional structure, all indices are accounted for.

The next example places two indices on each of the row and column labels and retains five decimal places:

option x:5:2:2; display x;

The output is:
--- 18 PARAMETER x a four dimensional structure

  a.i  a.ii  b.i  b.ii

first .one      5.63559
first .two  -2.93930  0.02873  10.34570
first .three  -6.31610
second.one    0.00013  1.00370 -17.29948
second.two  +INF  20000.00000

If in the option statement less dimensions are specified than the number of indices in the domain, then subtables will be generated for the labels of the beginning indices. Observe how the display layout will change if we use the following statement:

option x:0:1:1; display x;

The output follows.

--- 19 PARAMETER x a four dimensional structure

INDEX 1 = first  INDEX 2 = one
   i

   b  6

INDEX 1 = first  INDEX 2 = two
   i  ii

   a  -3  2.873000E-2
   b  10

INDEX 1 = first  INDEX 2 = three
   i

   b  -6

INDEX 1 = second  INDEX 2 = one
   i  ii

   a  1.300000E-4  1
   b  -17

INDEX 1 = second  INDEX 2 = two
   i  ii

   a  +INF
   b  20000

The option statement is checked for consistency against the dimensionality of the identifier and error messages will be issued if the number of the dimensions specified in the option statement is larger than the number of indices in the domain of the identifier.
4.32 The Display Statement

4.32.4.3 Display Statement to Generate Data in List Format

The option statement for local display controls may be used to generate data in list format by setting the \texttt{r-value} to zero:

\begin{verbatim}
option ident:d-value:0:c-value;
\end{verbatim}

In this case the \texttt{c-value} specifies the maximum number of items displayed on a line. The actual number will depend on the page width and the number and length of the labels.

Using the same example as in the previous sections, we change the \texttt{r-value} to zero in the option statement:

\begin{verbatim}
option x:5:0:1; display x;
option x:5:0:2; display x;
\end{verbatim}

Note that we specified one and two columns to illustrate the impact of different \texttt{c-value} settings. These statements will generate the following output:

\begin{verbatim}
---- 20 PARAMETER x a four dimensional structure
first .one .b.i  5.63559
first .two .a.i  -2.93930
first .two .a.ii  0.02873
first .two .b.ii  10.34570
first .three .b.i  -6.31610
second .one .a.i  0.00013
second .one .a.ii  1.00370
second .one .b.ii  -17.29948
second .two .a.ii  +INF
second .two .b.i  20000.00000

---- 21 PARAMETER x a four dimensional structure
first .one .b.i  5.63559, first .two .a.i  -2.93930
first .two .a.ii  0.02873, first .two .b.ii  10.34570
first .three .b.i  -6.31610, second .one .a.i  0.00013
second .one .a.ii  1.00370, second .one .b.ii  -17.29948
second .two .a.ii  +INF, second .two .b.i  20000.00000
\end{verbatim}

This output nicely illustrates the label order used. The first index varies the slowest, the last the fastest, and each one runs from beginning to end before the next one to the left advances. This ordering scheme is also used in equation and column lists and in the solution report which are all generated by the solve statement.
### 4.32.5 Conditional Displays

This section assumes familiarity with the if statement and the dollar condition.

The display statement may be controlled by conditional expressions. Conditionals have the effect that the items are displayed only if a logical condition is satisfied. There are two ways to express conditional displays in GAMS: with dollar conditions and using if statements. The syntax using a dollar condition is as follows:

\[
\text{display$\text{logical\_condition} \text{ ident | quoted text } \{, \text{ ident | quoted text}\};}
\]

Note that the dollar condition is inserted after the keyword display. For details on the various forms of conditional expressions in GAMS, see sections Logical Conditions and Filtering Sets. Consider the following simple example:

```gams
Scalars x, y;
x = 7;
y = 3;
display$(x-y < 0) "display if x minus y is less than 0", x, y;
```

The logical condition \((x-y < 0)\) controls the display statement. In this case it is not satisfied, hence there will be no display output. If the value of \(y\) is changed to, say 10, then the logical condition will be satisfied and the display statement will generate the following output:

```
---- 4 display if x minus y is less than 0
       PARAMETER x = 7.000
       PARAMETER y = 10.000
```

The syntax for conditional displays using an if statement is as follows:

```gams
if( logical\_condition, display\_statement );
```

The simple example above may be reformulated as follows:

```gams
Scalars x, y;
x = 7;
y = 3;
if( x-y < 0,
   display "display if x minus y is less than 0", x, y;
);
```

### 4.33 Programming Flow Control Features

#### 4.33.1 Introduction

In this chapter we will describe the programming flow control features available in GAMS. The if statement facilitates expressing complex conditional statements (see also chapter Conditional Expressions, Assignments and Equations).

In addition, GAMS offers four loop constructs to handle looping requirements: the loop statement, the while statement, the for statement and the repeat statement. At the end of this chapter we will introduce the break, and continue statements, which give additional control over the execution of loop structures, and the abort statement, a statement that may be used to terminate the execution of a program.

Note that this chapter deals with programming flow control features at execution time. In addition, GAMS provides a dollar control option that allows for conditional processing of input files at compile time. For more information, see the detailed description of the option $if. For details on dollar control options in general, see chapter Dollar Control Options.
4.33.2 The If Statement

The *if* statement is useful to branch conditionally around a group of statements. In some cases this can also be written as a set of dollar conditions, but the *if* statement may make the GAMS code more readable. An optional *else* and/or *elseif* part allows the user to formulate traditional *if-then-else* constructs.

4.33.2.1 The If Statement: Syntax

The syntax for an *if* statement in GAMS is as follows:

```gams
if (logical_condition,
    statement; {statement;}
{ elseif logical_condition,
    statement; {statement;}} 
[ else
    statement; {statement;] }
);
```

The keyword *if* indicates that this is an *if* statement. The logical condition is followed by one or more *statements* that are executed if the logical condition is satisfied. For details on the various forms of logical conditions in GAMS, see sections *Logical Conditions* and *Filtering Sets*. The *if* statement may end at this point, without any specifications for cases when the logical condition is *FALSE* and no action is taken in these cases. However, GAMS also allows further specifications: optional alternative *if* tests and optional *statements* for cases when all previous *if* tests have failed. The keyword *elseif* introduces an alternative *if* test with a logical condition and one or more *statements*. Note that the *elseif* case is optional and may be repeated multiple times. Note further that the *elseif* case implies that the logical condition of the *if* case has not been satisfied. The keyword *else* introduces the final part of the construct. It is optional and allows specification for cases when the logical condition of the *if* case (and possibly the *elseif* case(s)) has not been satisfied.

**Attention**

Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside an *if* statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option `onEnd`.

4.33.2.2 The If Statement: Examples

Consider the following set of conditional assignment statements that use dollar conditions:

```gams
p(i)$(f <= 0) = -1 ;
p(i)$(f > 0) and (f < 1)) = p(i)**2 ;
p(i)$(f >= 1) = p(i)**3 ;
q(j)$(f <= 0) = -1 ;
q(j)$(f > 0) and (f < 1)) = q(j)**2 ;
q(j)$(f >= 1) = q(j)**3 ;
```
They may be expressed using an if-elseif-else statement:

```gams
if (f <= 0,
    p(i) = -1 ;
    q(j) = -1 ;
elseif ((f > 0) and (f < 1)),
    p(i) = p(i)**2 ;
    q(j) = q(j)**2 ;
else
    p(i) = p(i)**3 ;
    q(j) = q(j)**3 ;
)
```

Note that the body of the if statement may contain solve statements. For instance, consider the bit of GAMS code that follows. Note that ml is a GAMS model, z is a free variable, j is a set and x is a variable.

```gams
solve ml using lp minimizing z;
if (ml.modelstat = 4,
    display "model ml was infeasible, relax bounds on x and solve again";
    x.up(j) = 2*x.up(j) ;
    solve ml using lp minimizing z ;
else
    if (ml.modelstat <> 1,
        abort "error solving model ml" ;
    );
);
```

First the model ml is solved. For details on solve statements in GAMS, see section The Solve Statement. Then a post solution analysis is done with the if statement. If the model is infeasible, the upper bound on the variable x is relaxed and the model is solved again. If the original model is not infeasible and it is not optimal either, then the compilation is aborted and the error message above is reported. For more information on GAMS output, see chapter GAMS Output, particularly subsection Model Status for a list of all GAMS model statuses. The display statement is introduced in chapter The Display Statement. For details on the abort command, see abort.

The following GAMS code is illegal since it is not permitted to define equations inside an if statement.

```gams
if (s > 0,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside an if statement are not allowed.

```gams
if (s > 0,
    scalar y ; y = 5 ;
);
```

### 4.33.3 The Loop Statement

The loop statement facilitates executing a group of statements for each member of a set. Loop statements are particularly useful for cases when parallel assignments are not sufficient. This is the case most often when there is no analytic relationship between the values to be assigned to a parameter. It is, of course, also useful to have a looping statement for general programming.
4.33.3.1 The Loop Statement: Syntax

The syntax for the loop statement in GAMS is as follows:

```
loop(index_list[$(logical_condition)],
    statement; {statement;}
) ;
```

The keyword loop indicates that this is a loop statement. The index_list is the controlling domain of the loop. Note that loops may be controlled by more than one set. In this case parentheses are required around the index list, which is also called the loop set(s). Observe that dynamic sets are allowed as loop sets. The loop set(s) may be restricted by a logical condition. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. For an introduction to dollar conditions in general, see chapter Conditional Expressions, Assignments and Equations. The index list is followed by one or more statements. Except for the final statement, each statement must end with a semicolon ';'. The loop statements are executed for each member of the controlling domain in turn. The order of evaluation is the entry order of the labels. A loop is thus another, more general, type of indexed operation.

Attention

- Only execution statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a loop statement.
- It is illegal to modify any controlling set inside the body of the loop.

Loop statements in GAMS are often used for iterative calculations, generating reports with put statements and doing scenario based studies with solve statements. Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.3.2 The Loop Statement: Examples

Consider a hypothetical case when a growth rate is empirical:

- Set t / 1985*1990 /
- Parameter pop(t) / 1985 3456 / 

The loop statement is then used to calculate the cumulative sums iteratively:

```
loop(t, pop(t+1) = pop(t) + growth(t) ) ;
```

In this example the driving set is t and we have just one statement in the scope of the loop.

The following small artificial examples illustrate the effects of a dollar condition in a loop statement.
Sets
  i / i1*i3 /
  j / j1*j5 /
  k(i,j) / i1.j1, i1.j3, i3.j3, i3.j5 /
Parameter
  c(i) / i1 3, i2 1 /
  q(i,j) / i1.j1 1, i1.j2 3, i1.j4 2 /
Scalars
  x, y, z;
  x = 1; y = 3; z = 1;
loop ( (i,j) $ (q(i,j) > 0), x = x + q(i,j));
loop ( i $ (c(i) + c(i)**2), z = z + 1);
loop ( i $ sum(j, abs(q(i,j))), z = z + 1);
loop ( j $ (ord(j) > 1 and ord(j) < card(j)), z = z + 1);
loop ( (i,j) $ k(i,j), y = y + ord(i) + 2*ord(j));

In the first loop statement the controlling domain is the set pair (i,j). It is restricted to those label combinations whose values associated with the parameter q are greater than zero. The logical condition in the second loop statement is c(i) + c(i)**2. This is shorthand for c(i) + c(i)**2 != 0. The domain in the third loop statement is restricted to those elements of the set i where the sum over j of the absolute values of the parameter q does not equal zero. Note that this condition is satisfied only for the label "i1". Observe that i and j are both ordered sets. In the fourth loop statement the first and the last element of the set j are excluded. For more on the set operators ord and card, see sections The Ord Operator and The Card Operator respectively. The dollar control in the last loop statement excludes all label combinations that are not members of the set k. For further details on dollar conditions, see chapter Conditional Expressions, Assignments and Equations.

Note
The dollar condition may be replaced by an if statement; see the example below.

The next example shows how a model can be solved for each element of a set i with different data using a loop statement. Note that problemdata is a scalar, g(i) and d(i) are parameters, mymodel is a GAMS model and profit is a free variable.

loop (i,  
  problemdata = g(i);  
  solve mymodel using lp maximizing profit;  
  d(i) = profit.l;  
);

In the first statement some data in the model is updated in accordance with the ith element of the parameter g. In the second statement the model is solved. For details on the solve statement, see section The Solve Statement. The objective value for each iteration is saved in the parameter d in the third statement.

A loop is often used to perform iterative calculations. Consider the following example, which finds square roots by Newton’s method. This example is purely for illustration - in practice, the function sqrt should be used. Newton’s method is based on the assertion that if x is an approximation to the square root of v, then (x + v/x)/2 is a better approximation.

Set
  i "set to drive iterations" / i-1*i-100 /
Parameter
  value(i) "used to hold successive approximations";
Scalars
  target "number whose square root is needed" / 23.456 /
  sqrtval "final approximation to sqrt(target)"
  curacc "accuracy of current approximation"
  reltol "required relative accuracy" / 1.0e-06 /;
4.33 Programming Flow Control Features

abort\$(target <= 0) "argument to newton must be positive";
value("i-1") = target/2 ;
curacc = 1 ;
loop(i$(curacc > reltol),
  value(i+1) = 0.5*(value(i) + target/value(i));
  sqrtval = value(i+1);
  curacc = abs (value(i+1)-value(i))/(1+abs(value(i+1)))
  ) ;
abort\$(curacc > reltol) "square root not found"
option decimals=8;
display "square root found within tolerance", sqrtval, value;

Note that in this example the dollar condition in the loop does not restrict the driving set \( i \), but it is used to terminate the loop procedure. The scalar \( \text{curacc} \) is updated in every iteration. As soon as it becomes equal to or smaller than the required relative accuracy \( \text{reltol} \) the loop stops. As the output below shows, this is the case after seven iterations. The body of the loop statement consists of three statements. The first statement calculates the current approximation and assigns it to the parameter \( \text{value} \). The second statement updates the scalar \( \text{sqrtval} \), and the third statements computes the accuracy of the current approximation in each iteration. Note that before and after the loop statement we added lines to account for special cases. For details on the abort statement, see section The Abort Statement at the end of this chapter. The output generated by the display statement is given below.

---

19 square root found within tolerance

---

19 PARAMETER SQRTVAL = 4.84313948 final approximation to sqrt(target)

---

19 PARAMETER VALUE used to hold successive approximations

i-1 11.72800000, i-2 6.86400000, i-3 5.14062471, i-4 4.85174713
i-5 4.84314711, i-6 4.84313948, i-7 4.84313948

Note that a statement within the body of a loop may be an if statement (or any other programming flow control statement). Moreover, the logical condition in a loop statement may be expressed with an if statement instead of a dollar condition. The following example serves as illustration. Observe that \( k \) is a set and \( s, t, u \) and \( a \) are parameters.

loop \( (k, \)
  if((s(k) < 0 and t(k)),
    u(k) = a(k);
  );
); 

Note that if the logical condition is not satisfied the assignment is not made and the parameter \( u \) remains unchanged.

Recall that subsets are connected with their supersets by arcs thus building a domain tree where the root node is the universal set. The following example demonstrates how the domain tree may be used in a loop statement.
Set i / i1*i10 /  
  ii(i)  
  j(i) / i1*i9 /  
  jj(j) / i1*i8 /  
  jjj(jj) / i1*i7 /;

loop(i(jjj), ii(i) = yes;);
display ii;

Observe that the looping set is i(jjj). This means that we loop over those elements of i that are also elements of the set jjj. This construct is permitted since i is in the domain tree on the path from jjj to the universe or universal set. It is allowed to go up the domain tree on one path and go down on another path. Therefore all the elements of jjj are assigned to ii. The outcome of the display statement confirms this:

---- 8 SET ii
i1, i2, i3, i4, i5, i6, i7

4.33.4 The While Statement

The while statement facilitates the repeated execution of one or more statements as long as a logical condition is satisfied.

4.33.4.1 The While Statement: Syntax

The syntax for the while statement in GAMS is as follows:

while(logical_condition,
    statement; {statement;}
);

The keyword while indicates that this is a while statement. Inside the while statement a logical condition is followed by one or more statements. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. The statements are executed as long as the logical condition is TRUE.

Attention

Only execution statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a while statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.
4.33.4.2 The While Statement: Examples

Consider the following simple example:

Scalar x; x = 1;
while ( round(x,2) < 10,
    x = x + 0.01;
);
display x;

Note that the scalar x is increased in each iteration until it equals 10. Note further, that to ensure an exact result, in numerical comparisons we need a stable check like we have above (round(x,2) < 10), otherwise rounding errors may occur. As soon as x reaches 10, the logical condition is no longer satisfied and therefore there will be no further passes. Hence the final value of x equals 10.

Note that the number of passes in a while statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

While statements may be used to control the solve statement. For instance, consider the following implementation of a random multi-start method for non-convex optimization.

scalar count / 1 /;
scalar globmin / inf /;
option bratio = 1;
while(count <= 1000,
    x.l(j) = uniform(x.lo(j),x.up(j)) ;
solve ml using nlp minimizing obj ;
if (obj.l < globmin,
    globmin = obj.l ;
    besstop(j) = x.l(j) ;
) ;
count = count+1 ;
);

Note that we start from a random starting point by setting the initial level values randomly between the upper and lower bounds. This assumes that the bounds have been previously specified and are not infinity. When the method improves, that is, if the logical condition (obj.l < globmin) is satisfied, the best known solution is stored in the scalar globmin. The level values associated with the best known solution so far are then saved in the parameter besstop. This procedure is repeated 1000 times. The model [PRIME] is another example where the use of the while statement is illustrated. In this model the set of all prime numbers smaller than 200 is generated.

The following GAMS code is illegal since equation definitions inside a while statement are not permitted.

while(s > 0,
    eq.. sum(i,x(i)) =g= 2 ;
);

The following GAMS code is also illegal since declarations inside a while statement are not allowed.

while(s > 0,
    scalar y ; y = 5 ;
);
4.33.5 The For Statement

The for statement provides a compact way to iterate over a range of values and execute one or more statements each time.

4.33.5.1 The For Statement: Syntax

The syntax for the for statement in GAMS is as follows:

```gams
for (a = start_value to|downto end_value [by incr],
    statement; {statement;}
); 
```

The keyword for indicates that this is a for statement. The scalar a begins with the real number start_value and is changed after each pass of the loop by the increment incr until it reaches the real number end_value. Note that the specification of an increment is optional, the default is 1. If the increment is given, it has to be a positive real number. Note further that to indicates that the scalar a is increased and downto indicates that it is decreased. In each iteration one or more statements are executed.

Attention

Only execution statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a for statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.5.2 The For Statement: Examples

Consider the following simple example:

```gams
Scalar s;
for (s = -3.8 to -0.1 by 1.4,
    display s ;
); 
```

Note that negative real numbers are possible for the start and end values. The resulting listing file will contain the following lines:

```
---- 3 PARAMETER s = -3.800
---- 3 PARAMETER s = -2.400
---- 3 PARAMETER s = -1.000
```

Observe that the value of s was increased by 1.4 with each pass of the loop as long as it did not exceed -0.1. In the next example the value of s is decreased:
Scalar s;
for (s = 3 downto -0.1 by 1.4,
     display s ;
);

Note that the number of passes in a for statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

Like while statements, for statements may be used to control the solve statement. The following example illustrates the use of the for statement by replicating the random search for a global optimum of a non-convex model that we discussed above.

scalar i ;
scalar globmin / inf / ;
option bratio = 1 ;
for (i = 1 to 1000,
     x.l(j) = uniform(x.lo(j),x.up(j)) ;
     solve ml using nlp minimizing obj ;
     if (obj.l < globmin,
         globmin = obj.l ;
         bestsol(j) = x.l(j) ;
     );
);

Note that the logical condition in the while loop (count <= 1000) is replaced by the specification of the range of values for the scalar i. The body of the for loop is identical to the body of the while loop, except for the statement to update the scalar count that we needed in the while loop. This demonstrates the similarities and differences between the two loops.

The following GAMS code is illegal since it is not allowed to define equations inside a for statement.

for (s = 1 to 5,
     eq.. sum(i,x(i)) =g= 2 ;
);

The following GAMS code is also illegal since declarations inside a for statement are not permitted.

for (s = 1 to 5,
     scalar y ; y = 5 ;
);

4.33.6 The Repeat Statement

The repeat statement facilitates the repeated execution of one or more statements. This is done unconditionally at least once and stopped when a logical condition is satisfied.
4.33.6.1 The Repeat Statement: Syntax

The syntax for the repeat statement in GAMS is as follows:

```gams
repeat (
    statement; {statement;}
until logical_condition );
```

The keyword repeat indicates that this is a repeat statement. One or more statements are executed in each iteration. The keyword until introduces the termination criterion: if the logical condition is satisfied, the repeat loop is terminated. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

Note that the repeat statement is similar to the while statement, but a repeat loop is guaranteed to be executed at least one time since the logical condition is stated after the statements.

Attention

Only execution statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a repeat statement.

Examples are given in the next subsection.

4.33.6.2 The Repeat Statement: Examples

Consider the following simple example:

```gams
Scalar a / 1 /;
repeat (
    a = a + 1;
    display a;
until a = 5 );
```

The scalar `a` is increased in each iteration by 1. If `a` equals 5, the termination criterion is satisfied and the loop stops. Note that this example works nicely since both, the scalar and the increment, are integer. In case the entity on the right-hand side of the termination condition is not an integer or the increment is not an integer, we recommend a formulation of the check that is more stable to avoid rounding errors. An example for a stable termination condition follows.

```gams
Scalar a / 1 /;
repeat (
    a = a + 0.1;
    display a;
until abs(a-5) < 1e-6 );
```

Observe that in the next example the termination condition is TRUE from the start. In this case the statement in the body of the repeat statement is executed once and then the loop is terminated. Hence the final value of `a` will be 5.
Scalar a / 4 /;
repeat (  
a = a + 1;
display a;
until a >= 3 );

Note that the number of passes in a while statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

Here is a little more complex example. A repeat statement is used to narrow the interval where a quadratic function passes through zero. Note that, as is often the case, one of the statements in the repeat loop is another programming flow control statement, in this case an if statement.

Scalar max "current upper boundary of interval" /10/  
min "current lower boundary of interval" /-10/  
root "value where function equals zero"  
function_value1 "function value at min"  
function_value2 "function value at max"  
tolerance "tolerance for root" /0.00000001/  
signswitch "indicates that sign switch was found" /0/  
inc "increment to try to find sign switch" ;

function_value1 = 6 - 5*min + sqr(min);
inc = (max - min)/37;
root = min;
repeat (  
    root = root + inc;
    function_value2 = 6 - 5*root + sqr(root);
    if(( sign(function_value1) <> sign(function_value2)  
        and abs(function_value1) > 0  
        and abs(function_value2) > tolerance ),
        max = root;
        signswitch = 1;
    else
        if(abs(function_value2) > tolerance,  
            function_value1 = function_value2;
            min = root;
        );
    );
until (signswitch > 0) or (root > max );
display min, max, function_value1, function_value2;

The result of the display statement shows that the value of min is 1.892 and the value of max is 2.432, the interval was narrowed to 0.54, which is just a little less than the value of inc. As expected, function_value1 and function_value2 differ in sign, confirming that the root of the quadratic function is indeed in the interval.

4.33.7 The Break Statement

The break statement gives additional control over the execution of loop structures, namely the loop statement, the while statement, the for statement and the repeat statement. It allows to break the execution of a loop structure prematurely.
4.33.7.1 The Break Statement: Syntax

The syntax for the break statement is as follows:

break[n];

The keyword break indicates that this is a break statement. It terminates the n inner most control structures. n is optional and if it is omitted, it is set to 1.

Most often break statements are used in the context of if statements or with dollar conditions. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.7.2 The Break Statement: Examples

This is a simple, artificial example using the break statement to exit a loop statement:

Set i / i1*i10 /;
Scalar cnt / 0 /;
loop(i,  
  break$sameas('i6',i);  
  cnt = cnt+1;  
);
display cnt;

The break statement in combination with the dollar condition terminates the execution of the loop body in the 6th iteration. This can be seen looking at the value of cnt after the loop, which will be 5.

Here is a little more complex example which uses the optional argument n to terminate more than one loop structure at once:

Set i / i1*i10 /;
Scalar x, y,  
cnt / 0 /;
for(x = 1 to 10,  
  y = 0;  
  while(y < 10,  
    y = y+1;  
    loop(i,  
      break$sameas('i6',i) 2;  
      cnt = cnt+1;  
    );  
  );  
);
display cnt;

In this example, cnt will be 50 at the end. As in the previous example, it is increased 5 times in the inner most loop, before the break statement is executed. This time break is called with the argument 2, which causes the two inner most control structures (namely the loop and the while) to be terminated. The outer most control structure (the for) is not influenced, so that its body gets executed 10 times, which results in a total of 50 increments for the scalar cnt.

Note, if break would be called with the argument 3 instead of 2, also the for loop would be terminated, so that cnt would be 5 at the end. If break would be called with 1 (or without additional argument) instead of 2, only the inner most loop would be terminated so that cnt would be 500 at the end.
4.33 Programming Flow Control Features

4.33.8 The Continue Statement

The continue statement gives additional control over the execution of loop structures, namely the loop statement, the while statement, the for statement and the repeat statement. It allows to jump to the end of the inner most loop structure without executing the remaining statements in the body.

4.33.8.1 The Continue Statement: Syntax

The syntax for the continue statement is as follows:

```
continue;
```

The keyword continue indicates that this is a continue statement. It jumps to the end of the inner most control structure.

Most often continue statements are used in the context of if statements or with dollar conditions. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.8.2 The Continue Statement: Examples

This is a simple, artificial example using the continue statement to skip parts of a loop body:

```
Set    i / i1*i10 /;
Scalar cnt / 0 /;
loop(i,
    continue$(mod(ord(i),2)=0);
    cnt=cnt+1
);
display cnt;
```

In that example, every 2nd iteration of the loop statement is skipped. Therefore, cnt will be 5 at the end of the loop.

4.33.9 The Abort Statement

The abort statement is used to terminate the execution of a program. Most often the abort statement is used in the context of conditionals. Examples are given below. Also it may be used to display a text or an identifier in the listing file similar to the display statement but mostly to present the reason for the termination of the execution.

Note that the abort statement is to be distinguished from the dollar control option $abort which may be used to terminate the compilation of a program.
4.33.9.1 The Abort Statement: Syntax

The syntax for the `abort` statement is as follows:

```plaintext
abort [ ident | quoted text {, ident | quoted text} ];
```

The keyword `abort` indicates that this is an abort statement, all other elements of the statement are optional. `Ident` denotes an identifier. If the identifier is a set or a parameter, only the name of the set or parameter itself is specified, without any domains. If the identifier is a variable, an equation or a model, it must be followed by a suffix, since only `attributes` of variables, equations and models can be displayed. For more on variable and equation attributes including full lists, see sections Variable Attributes and Equation Attributes respectively. For details on model attributes, see section Model Attributes. Recall that sets also have attributes, they may also be displayed using the suffix notation. For details on set attributes, see section Set Attributes. As usual, `quoted text` must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines.

An abort statement causes the termination of the execution with an execution error and the information in the statement will be displayed.

There is also a variant with the extension `.noError` that terminates the execution and displays the information, but does not cause an execution error. The syntax is as follows:

```plaintext
abort.noError [ ident | quoted text {, ident | quoted text} ];
```

Most often `abort` statements are used in the context of `if` statements or with `dollar conditions`. The syntax is as follows:

```plaintext
if (logical_condition, abort [ ident | quoted text {, ident | quoted text} ]; );
```

or

```plaintext
abort$logical_condition [ ident | quoted text {, ident | quoted text} ];
```

For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.9.2 The Abort Statement: Examples

Consider the following artificial example:

```plaintext
Set i / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,  
  if (p(i) > 3, abort "Parameter larger than 3", p);  
  p(i) = p(i) + 2;
);
```
Note that the abort statement is part of an if statement which is part of a loop statement. The execution of this program will be terminated with the following display and error message:

```
---- 4 Parameter larger than 3
---- 4 PARAMETER p
i1 3.000, i2 4.000, i3 5.000, i4 5.000, i5 8.000
**** Exec Error at line 4: Execution halted: abort 'Parameter larger than 3'
```

Observe that the values of $p('1')$, $p('2')$ and $p('3')$ were updated, but the values of $p('4')$ and $p('5')$ are equal to the initialization values since the program was terminated before they could be updated.

If the extension .noError is used in the example above, the following lines will appear in the listing file:

```
---- 4 Parameter larger than 3
---- 4 PARAMETER p
i1 3.000, i2 4.000, i3 5.000, i4 5.000, i5 8.000
****
**** Execution halted from line 4: abort.noError 'Parameter larger than 3'
****
```

Note that the execution of the program is aborted as before, but there is no execution error in this case.

Instead of an if statement we may use a dollar condition in the loop:

```gams
Set i / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,
   abort$(p(i) > 3) "Parameter larger than 3", p;
   p(i) = p(i) + 2;
);
```

Observe that this alternative formulation has the same outcome as above.

### 4.34 The Option Statement

#### 4.34.1 Introduction

The option statement is used to set various global system parameters to control among other things output detail, the solution process and the layout of displays. GAMS provides default values for global system parameters that are adequate for the most purposes. However, there are always cases when the user would like to maintain control of some aspects of the run. In addition, the option statement provides an efficient and compact syntax to perform powerful operations on identifiers. Observe that option statements are processed at execution time unlike dollar control options that are processed at compile time.

This chapter is organized as follows. We will first introduce the general syntax of the option statement, then we will continue with a list of all available options that may be used with option statements and provide links to their detailed explanations in the GAMS Call chapter. Finally we turn to the special options that involve identifiers.
4.34.1.1 The Syntax of the Option Statement

The general form of an option statement is as follows:

```
option(s) key1 [= value1] { ,|EOL key2 [= value2] } ;
```

The keyword `option` or `options` indicates that this is an option statement. It is followed by `key1`, which is one of the options that are listed in this chapter. Consider the following simple example:

```
option reslim=800;
```

Here the keyword `option` is followed by the `key reslim`. The option `reslim` specifies the maximum time in seconds that a solver may run before it terminates. Thus, in this example we give the solver 800 seconds to come up with a solution.

Note

Option names are not reserved words and therefore they do not conflict with other uses of their names.

Observe that it depends on the respective option whether a value is expected and if so, what type of value. There are six different cases. An overview is given in Table 1.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Type of Value</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>no</td>
<td>-</td>
<td>dmpOpt, eject, memoryStat</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>integer number</td>
<td>decimals, limcol, seed</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>real number</td>
<td>FDDelta, optCR, resLim</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>text string</td>
<td>LP, solprint, sysout</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>identifier</td>
<td>See identifier options.</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>-</td>
<td>See identifier operations.</td>
</tr>
</tbody>
</table>

**Table 1: Types of Options**

Note that the last type is special: it does not involve a named option or key, but the keyword `option` is followed by identifiers and identifier operators. These special option statements are discussed in detail in section Special Options: Identifier Operations.

Observe that the value of an option may be reset as often as necessary, the new value will replace the previous value each time. Further, more than one option may be specified with one option statement and commas or end-of-line characters are both legal separators between options.

We will demonstrate with the following example how several options may be used. The code snippet may be added to the model [DICE].

```
option measure, limcol = 100
    optcr = 0.00, mip = xpress 
solve xdice using mip max wnx;
option clear = comp;
```
Note that in the first option statement four options are specified: option measure has no associated value, option limcol expects an integer value, option optcr expects a real value and option MIP expects a text string as value. The second option statement specifies just one identifier option: clear, which has the variable comp as value.

Attention

Option statements are executed in sequence with other instructions. Therefore, if an option statement is located between two solve statements, the new values will be assigned between the solves and thus they will apply only to the second solve statement.

4.34.2 List of Options

The options available through the option statement are grouped into the following functional categories:

- Options that affect output details
- Solver specific parameters
- Options that control choice of solver
- Options that affect input program control
- Other options
- Special options that involve identifiers

In the following subsections we will offer brief descriptions of the options in the first five categories. Note that each entry is linked to a detailed description of the respective option. Observe that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section Detailed Descriptions of All Options. The options that belong to the last category are special, they are introduced and discussed in section Special Options that Involve Identifiers below.

4.34.2.1 Options that Control Output Details

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asyncSolLst</td>
<td>Print solution listing when asynchronous solve (Grid or Threads) is used</td>
</tr>
<tr>
<td>decimals</td>
<td>Decimal places for display statements</td>
</tr>
<tr>
<td>dispWidth</td>
<td>Number of characters to be printed in the column labels of all subsequent display statements</td>
</tr>
<tr>
<td>eject</td>
<td>Inject a page break into the LST file</td>
</tr>
<tr>
<td>epsToZero</td>
<td>Treat eps as zero when unloading to GDX</td>
</tr>
<tr>
<td>limCol</td>
<td>Maximum number of columns listed in one variable block</td>
</tr>
<tr>
<td>limRow</td>
<td>Maximum number of rows listed in one equation block</td>
</tr>
<tr>
<td>maxGenericFiles</td>
<td>Maximum number of generic file names tried at execution time file creation</td>
</tr>
<tr>
<td>MCPRHoldFx</td>
<td>Print list of rows that are perpendicular to variables removed due to the holdfixed setting</td>
</tr>
<tr>
<td>profile</td>
<td>Execution profiling</td>
</tr>
<tr>
<td>profileTol</td>
<td>Minimum time a statement must use to appear in profile generated output</td>
</tr>
<tr>
<td>solPrint</td>
<td>Solution report print option</td>
</tr>
<tr>
<td>solSlack</td>
<td>Causes the equation output in the listing file to contain slack variable values instead of level values</td>
</tr>
<tr>
<td>sysOut</td>
<td>Solver Status file reporting option</td>
</tr>
</tbody>
</table>
4.34.2.2 Options that Control Solver-Specific Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRatio</td>
<td>Basis detection threshold</td>
</tr>
<tr>
<td>domLim</td>
<td>Domain violation limit solver default</td>
</tr>
<tr>
<td>holdFixedAsync</td>
<td>Allow HoldFixed for models solved asynchronously as well</td>
</tr>
<tr>
<td>intVarUp</td>
<td>Set mode for default upper bounds on integer variables</td>
</tr>
<tr>
<td>iterLim</td>
<td>Iteration limit of solver</td>
</tr>
<tr>
<td>optCA</td>
<td>Absolute Optimality criterion solver default</td>
</tr>
<tr>
<td>optCR</td>
<td>Relative Optimality criterion solver default</td>
</tr>
<tr>
<td>reform</td>
<td>Reformulation level</td>
</tr>
<tr>
<td>resLim</td>
<td>Wall-clock time limit for solver</td>
</tr>
<tr>
<td>savePoint</td>
<td>Save solver point in GDX file</td>
</tr>
<tr>
<td>solveLink</td>
<td>Solver link option</td>
</tr>
<tr>
<td>sys12</td>
<td>Pass model with generation errors to solver</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads to be used by a solver</td>
</tr>
</tbody>
</table>

4.34.2.3 Options that Control the Choice of Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS</td>
<td>Constrained Nonlinear Systems - default solver</td>
</tr>
<tr>
<td>DNLP</td>
<td>Non-Linear Programming with Discontinuous Derivatives - default solver</td>
</tr>
<tr>
<td>EMP</td>
<td>Extended Mathematical Programs - default solver</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming - default solver</td>
</tr>
<tr>
<td>MCP</td>
<td>Mixed Complementarity Problems - default solver</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed-Integer Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed-Integer Programming - default solver</td>
</tr>
<tr>
<td>MIQCP</td>
<td>Mixed Integer Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>MPEC</td>
<td>Mathematical Programs with Equilibrium Constraints - default solver</td>
</tr>
<tr>
<td>NLP</td>
<td>Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>QCP</td>
<td>Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>RMINLP</td>
<td>Relaxed Mixed-Integer Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>RMIP</td>
<td>Relaxed Mixed-Integer Programming - default solver</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>Relaxed Mixed Integer Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>RMPEC</td>
<td>Relaxed Mathematical Programs with Equilibrium Constraints - default solver</td>
</tr>
<tr>
<td>solver</td>
<td>Default solver for all model types that the solver is capable to process</td>
</tr>
</tbody>
</table>

4.34.2.4 Options that Affect Input Program Control

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECImplicitLoad</td>
<td>Allow implicit loading of symbols from embedded code or not</td>
</tr>
<tr>
<td>fdDelta</td>
<td>Step size for finite differences</td>
</tr>
<tr>
<td>fdOpt</td>
<td>Options for finite differences</td>
</tr>
<tr>
<td>gdxUels</td>
<td>Unload labels or UELs to GDX either squeezed or full</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed</td>
</tr>
<tr>
<td>solveOpt</td>
<td>Multiple solve management</td>
</tr>
<tr>
<td>strictSingleton</td>
<td>Error if assignment to singleton set has multiple elements</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sys18</td>
<td>Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers</td>
</tr>
<tr>
<td>zeroToEps</td>
<td>Treat zero as eps</td>
</tr>
</tbody>
</table>

### 4.34.2.5 Other Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkErrorLevel</td>
<td>Check errorLevel automatically after executing external program</td>
</tr>
<tr>
<td>dmpOpt</td>
<td>Debugging option: causes GAMS to echo the runtime option settings</td>
</tr>
<tr>
<td>dmpSym</td>
<td>Debugging option: causes GAMS to echo the symbol table to the listing file</td>
</tr>
<tr>
<td>dmpUserSym</td>
<td>Debugging option: causes GAMS to echo the symbol table to the listing file for user defined symbols only</td>
</tr>
<tr>
<td>dualCheck</td>
<td>Output on the reduced cost condition</td>
</tr>
<tr>
<td>forLim</td>
<td>GAMS looping limit</td>
</tr>
<tr>
<td>integer1..5</td>
<td>Integer communication cells</td>
</tr>
<tr>
<td>measure</td>
<td>Output of time and memory use since the last measure statement or the program beginning</td>
</tr>
<tr>
<td>memoryStat</td>
<td>Show memory statistics in the LST file</td>
</tr>
<tr>
<td>real1..5</td>
<td>Real communication cells</td>
</tr>
<tr>
<td>subSystems</td>
<td>Lists all solvers available as well as the current default and active solvers in the LST file</td>
</tr>
<tr>
<td>sys10</td>
<td>Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer</td>
</tr>
<tr>
<td>sys11</td>
<td>Dynamic resorting if indices in assignment/data statements are not in natural order</td>
</tr>
<tr>
<td>sys15</td>
<td>Automatic switching of data structures used in search records</td>
</tr>
<tr>
<td>sys16</td>
<td>Disable search record memory (aka execute this as pre-GAMS 24.5)</td>
</tr>
<tr>
<td>sys17</td>
<td>Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)</td>
</tr>
<tr>
<td>sys19</td>
<td>Disable permutation on Column Generation (aka execute this as pre-GAMS 36)</td>
</tr>
<tr>
<td>threadsAsync</td>
<td>Limit on number of threads to be used for asynchronous solves (solveLink=6)</td>
</tr>
</tbody>
</table>

### 4.34.2.6 Special Options that Involve Identifiers

Several options involve identifiers: they either take identifiers as values or have no key and value, but perform an operation on an identifier. In the following two subsections we will introduce and discuss these special options.

#### 4.34.2.6.1 Special Options: Identifier Options

The value of identifier options is not a string or a number, but an identifier. In this subsection we will describe these options in detail.

**clear**

This option resets an identifier to its default value to free memory. The syntax is as follows:

```
option clear = identifier;
```
The following identifier types may be reset: sets, parameters, equations and variables. The option will free up memory to the GAMS heap manager, thus the memory may be used by GAMS but not by the operating system. To force the memory to be freed up to the operating system, the GAMS process needs to be terminated. One way to do this is to restart the execution system by solving a dummy model with the option SolveLink set to zero.

**kill**

This option is a synonym to option clear. Observe that the dollar control option $kill is not a synonym to $clear.

**shuffle**

This option rearranges the values of a parameter in a random order. The syntax is as follows:

```plaintext
option shuffle = itemname;
```

Here `itemname` is a one-dimensional parameter. One-dimensional parameters may be declared (and defined) in four different ways, depending on the domain and the data. The following table gives an overview of the effect of the option `shuffle` in the four cases.

<table>
<thead>
<tr>
<th>Universal set as domain</th>
<th>Has Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the universal set to initialize the data (case 1 in the example below).</td>
<td>Use the universal set to add zero values before shuffling the data (case 3 in the example below).</td>
</tr>
<tr>
<td>Specific set as domain</td>
<td>Use the domain to initialize the data (case 2 in the example below).</td>
</tr>
</tbody>
</table>

If the parameter was declared without data (the second column in the table above), the domain or the universal set will be used to assign the numbers 1 to \( N \), where \( N \) is the number of elements in the domain or the universal set. If the parameter was declared with data (the third column in the table above), the domain or the universal set will be used to add zeroes for possibly missing entries. These zero values will participate in the random shuffle, but they will not be stored in the parameter. The following example serves as illustration:

```plaintext
Set i / i1*i5 /  
j / j1*j5 /;
option decimals = 0;

*Case 1: universal set as domain and no data
Parameter A(*) "The universe is used to fill the parameter";
option shuffle = A;
display A;

*Case 2: set j as domain and no data
Parameter B(j) "The set j is used to fill the parameter";
option shuffle = B;
display B;

*Case 3: universal set as domain and has data
Parameter C(*) "The universe is used to add zeroes" / j2 2, j4 4 /;
option shuffle = C;
display C;
```
The code above will generate the output that follows. Observe that in this example, the universal set is the union of the sets $i$ and $j$, which means that all elements of the sets $i$ and $j$ are members of the universal set. Note that we use random numbers, therefore the outcomes from a different run may vary.

---- 9 PARAMETER A  The universe is used to fill the parameter
i1 4,  i2 1,  i3 7,  i4 9,  i5 6,  j1 10,  j2 3,  j3 5
j4 8,  j5 2

---- 14 PARAMETER B  The set $j$ is used to fill the parameter
j1 1,  j2 5,  j3 2,  j4 4,  j5 3

---- 19 PARAMETER C  The universe is used to add zeroes
j1 2,  j2 4

---- 24 PARAMETER D  The set $i$ is used to add zeroes
i2 30,  i4 50,  i5 10

In the next example we will demonstrate how to generate a random mapping of a set:

Set $i$  / i1*i6 /,
    rmi(i,i) "random mapping";
Parameter A(i);
option shuffle = A;

rmi(i, i + (- Ord(i) + A(i))) = yes;
display rmi;

A display of the set $rmi$ follows. Note that there is exactly one element in each row and each column:

---- 7 SET rmi  random mapping

i1  i2  i3  i4  i5  i6
i1  YES
i2  YES
i3  YES
i4  YES
i5  YES
i6  YES

Observe that each use of the option shuffle will generate a new random data rearrangement.
4.34.2.6.2 Special Options: Identifier Operations  In some cases the keyword option in an option statement is followed by an identifier and one or more operators to perform identifier operations like display control, index matching, projection and aggregation of sets and parameters and permutation of sets and parameters.

4.34.2.6.2.1 Display Control  The display statement is introduced and discussed in chapter The Display Statement. While GAMS provides defaults for the displayed identifiers that suffice in most cases, the print format may be customized with the following option statement:

```plaintext
option ident:d;
option ident:d:r:c;
```

The keyword option is followed by the name of an identifier ident, a colon and an integer value d. Note that d may be between 0 and 8 and specifies the number of decimal places that will be displayed for the respective identifier. The specifications r and c are optional. They denote the number of index positions printed as row labels and the number of index positions printed as column labels respectively. Note that if r is zero, a list format will be used. For more information and examples, see sections Local Display Control and Display Statement to Generate Data in List Format.

4.34.2.6.2.2 Index Matching  Index matching is a very compact way to define multi-dimensional sets. The general syntax is as follows:

```plaintext
option set_name(index1:index2[:index3:...]);
```

The keyword option is followed by the name of the set, set_name, and two (or more) indices in parentheses that are linked with the matching operator ':'. Note that the set must have been declared earlier in the program. If the set has also been defined earlier in the program, it will be cleared first and then the matching operation will be processed. Note furthermore that the list of identifiers may be expanded to more than two and that besides the matching operator ':' also ',' may be used and will be interpreted as product operator. Consider the following example which also makes use of display control:

```plaintext
Set i / i1,i2/  
  j / j1,j2,j3 /  
  k / k1*k5 /  
  ij(i,j), ijk(i,j,k), kij(k,i,j);
* index matching
Option ij(i:j), ijk(i,j:k), kij(k:i,j);
* display control
Option ij:0:0:1, ijk:0:0:1, kij:0:0:1;
Display ij, ijk, kij;
```

In its simplest form the matching operator is used to create the two dimensional set ij.

```plaintext
-----     9 SET ij
  i1.j1
  i2.j2
```

Note that each member of the index \( i \) has been matched with a member of the index \( j \) until one of the indices ran out of members.

The index matching operations to define the three-dimensional sets \( ijk \) and \( kij \) illustrate a more sophisticated usage of the index operator. The sets \( ijk \) and \( kij \) are built with the indices from left to right using the product operator when a ',' is encountered or the matching operator when a '::' is found.

For \( ijk \) the first operator is the ',' which is interpreted as product operator for the sets \( i \) and \( j \) and hence builds the Cartesian product of the two sets which has six 2-tuples as elements (\( i1.j1, i1.j2, i1.j3, i2.j1, i2.j2, i2.j3 \)). The matching operator '::' is then applied to match those 2-tuples with the five set elements in \( k \). The resulting sets are:

---- 9 SET ijk

\[
\begin{align*}
i1.j1.k1 \\
i1.j2.k2 \\
i1.j3.k3 \\
i2.j1.k4 \\
i2.j2.k5 
\end{align*}
\]

---- 9 SET kij

\[
\begin{align*}
k1.i1.j1 \\
k1.i1.j2 \\
k1.i1.j3 \\
k2.i2.j1 \\
k2.i2.j2 \\
k2.i2.j3 
\end{align*}
\]

The previous example can be extended to define sets of even higher dimension in the following way:

```plaintext
set h / h1*h4 / 
  hijk_1(h:ijk); 
  hijk_2(h:i,j:k); 
* display control 
  Option hijk_1:0:0:1, hijk_2:0:0:1; 
  Display hijk_1, hijk_2;
```

Note that sets \( hijk_1 \) and \( hijk_2 \) will be different even though \( hijk_1 \) uses set \( ijk \) and \( hijk_2 \) uses the same matching and product operation used at definition of \( ijk \) but spelled out. As already mentioned above, the matching operator builds up the sets with the indices from left to right. Hence, as set \( ijk \) is build first and then used on the right of the matching operator the two sets are built up differently.

---- 19 SET hijk_1

\[
\begin{align*}
h1.i1.j1.k1 \\
h2.i1.j2.k2 \\
h3.i1.j3.k3 \\
h4.i2.j1.k4 
\end{align*}
\]
4.34.2.6.2.3 Projection and Aggregation of Sets and Parameters  In GAMS, projection and aggregation operations on sets may be performed in two different ways: with an assignment and the `sum` operator, and with an option statement.

Using an assignment and the `sum` operator is the slower but more intuitive way. Assignments and the `sum` operator are introduced and discussed in detail in chapter Data Manipulations with Parameters and section Indexed Operations respectively. Here we only show how they may be used in the context of sets to perform projections and aggregations. The following example serves as illustration.

Sets i        / i1*i3 /  
  j        / j1*j2 /  
  k        / k1*k4 /  
  ijk(i,j,k) / #i.#j.#k /  
  ij1a(i,j)  
  ij1b(i,j);  

Scalars Count_1a, Count_1b, Count_2a, Count_2b;  

* Method 1: Using an assignment and the sum operator for a projection  
ij1a(i,j) = sum(k, ijk(i,j,k));  

* Method 1: Using an assignment and the sum operator for aggregations  
Count_2a = sum(ijk(i,j,k),1);  
Count_1a = sum(ij1a(i,j),1);  

Note that the set `ijk` is a three-dimensional set, its elements are 3-tuples and all permutations of the elements of the three sets `i`, `j` and `k` are in its domain. Thus the number of elements of the set `ijk` is 3 x 2 x 4 = 24. The sets `ij1a` and `ij1b` are two-dimensional sets that are declared in the set statement, but not defined. The first assignment statement defines the members of the set `ij1a`. This is a projection from the set `ijk` to the set `ij1a` where the three-tuples of the first set are mapped onto the pairs of the second set, such that the dimension `k` is eliminated. This means that the four elements "i1.j1.k1", "i1.j1.k2", "i1.j1.k3" and "i1.j1.k4" of the set `ijk` are all mapped to the element "i1.j1" of the set `ij1a`. Note that in this context, the result of the `sum` operation is not a number but a set. The second and third assignments are aggregations, where the number of elements of the two sets are computed. As already mentioned, the result of the first aggregation is 24 and the result of the second aggregation is 6 = 24 / 4.

The second way to perform projections and aggregations is faster and more compact, it uses an option statement. The general syntax of this option statement is as follows.

```gams
option ident1 < ident2 ;
option ident1 <= ident2 ;
```
The keyword *option* is followed by the identifiers *ident1* and *ident2* which are linked by the symbol `'<'` or `'<='`. Observe that in most cases the two symbols have the same effect. The exception is the special case when both identifiers are defined over domains that use at least one shared index set more than once, see the example below. Note that in general the dimension of the item on the left has to be equal or less than the dimension of the item on the right. Further, the index space of the two identifiers must be matchable. If the dimensions of the two identifiers are equal, then the same indices must appear in both, albeit the order may differ. If the dimension of the left item is less than the dimension of the right item, then the indices on the left must also appear on the right.

Observe that if both identifiers are sets, the operation will be a *projection*. However, if the identifier on the left-hand side is a *scalar* or a *parameter* and the identifier on the right-hand side is a set it will be an *aggregation*. The example that follows shows how the projection and the two aggregations above are accomplished with the option statement.

* Method 2: Option statement performs a projection
  Option ij1b < ijk;

* Method 2: Option statements performs aggregations (counting of elements)
  Option Count_2b < ijk;
  Option Count_1b < ij1b;

display ijk, ij1a, ij1b, Count_1a, Count_1b, Count_2a, Count_2b;

In the example above, the set on the left-hand side, *ij1b*, has fewer indices than the set on the right-hand side, *ijk*. Observe that if the two sets differ only in the order of the indices then a projection will have the effect of a *permutation* of the tuple.

Note

The option statement for projection and aggregation operations may also be applied to parameters.

Until now the indices in the domain of the sets were unique. A special case arises when sets are defined over a domain with the same indices, for example the set *s(i,i,i)*. In this case, a projection always has the effect of a permutation. Users may choose whether they wish to perform the permutation from left to right or from right to left. The option statement

Option item1 < item2;

means a right-to-left permutation, while the option statement

Option item1 <= item2;

entails a left-to-right permutation. The following example clarifies the difference:

Set i / i1*i3 /
s(i,i,i) "Set members" / i1.i2.i3, i3.i3.i1/
pR1(i,i) "projection right to left with assignment"
pR2(i,i) "projection right to left with option statement"
pL1(i,i) "projection left to right with assignment"
pL2(i,i) "projection left to right with option statement";
Alias (i,ii,iii);

* Right-to-left permutation, two ways
pR1(i,ii) = sum(s(iii,ii,i),1);
option pR2 < s;

* Left-to-right permutation, two ways
pL1(i,ii) = sum(s(i,ii,iii),1);
option pL2 <= s;

option s:0:0:1, pR1:0:0:1, pR2:0:0:1, pL1:0:0:1, pL2:0:0:1;
display s, pR1, pR2, pL1, pL2 ;

Note that in the right-to-left permutation, the element "i1.i2.i3" is projected to "i3.i2" and the element "i3.i3.i1" is projected to "i1.i3". In the left-to-right permutation however, the element "i1.i2.i3" is projected to "i1.i2" and the element "i3.i3.i1" is projected to "i3.i3". Hence, the left-to-right permutation (<=) might be more intuitive.

Our examples so far involved only sets. As mentioned above, projections and aggregations may also be performed with parameters. However, there are some subtle differences. The first difference refers to the terminology: we aggregate sets, but we count parameters. The second difference is the result of the operation if the domain of the left symbol is just a permuted version of the domain of the right symbol. Consider the following example:

Set i / i1*i3 /
   j / j1*j2 /;
Table p(i,j)
   j1 j2
  i1  1  2
  i2  3  4
  i3  5  6;
parameter pperm(j,i);
option pperm < p;
option decimals = 0;
display p, pperm;

The output generated by the display statement follows:

---- 13 PARAMETER p

   j1   j2
i1  1  2
i2  3  4
i3  5  6

---- 13 PARAMETER pperm

   i1   i2   i3
j1  1   3   5
j2  2   4   6
4.34 The Option Statement

4.34.2.6.2.4 Permutation of Sets and Parameters  In GAMS, the > sign can be used to create complete permutations of one- and multi-dimensional sets and parameters. Consider the following example based on the test library model [PERM1]:

First, a set \( i \) for which all permutations should be computed is declared and defined. A permutation of a one-dimensional set \( i \) can be represented by a two-dimensional set like \( \text{perm}(i,i) \) where the \( i \) index is duplicated. In the example, set \( \text{perm}(i,i) \) represents permutation \((1 2)(3)\) in cycle notation. For a set with three elements, there are \(3 \times 2 \times 1 = 6\) permutations. The statement \( \text{option pall} > i; \) results in all permutations of \( i \) being computed and stored in three-dimensional set \( \text{pall}(p,i,i) \) where the first index \( p \) serves as a counter to enumerate all permutations.

```gams
set i 'set to permute' / i1*i3 / 
perm(i,i) 'exemplary permutation' / i1.i2, i2.i1, i3.i3 / 
$eval pmax fact(card(i)) 
p 'permutation index' / p1*p%pmax% / 
pall(p,i,i) 'permutation set';
option pall > i;
option pall:0:0:1;
display pall;
```

Note that we use \( \text{display control} \) such that the display statement results in:

```
---- 8 SET pall permutation set

p1.i1.i1
p1.i2.i2
p1.i3.i3
p2.i1.i1
p2.i2.i3
p2.i3.i2
p3.i1.i2
p3.i2.i1
p3.i3.i3
p4.i1.i2
p4.i2.i3
p4.i3.i1
p5.i1.i3
p5.i2.i1
p5.i3.i2
p6.i1.i3
p6.i2.i2
p6.i3.i1
```

For multi-dimensional sets, the permutation operator follows the same logic. Consider the following example:

Sets \( j, k \) and a two-dimensional set \( \text{jk}(j,k) \) are defined. In order to find all permutations of \( \text{jk}(j,k) \), the indices are again duplicated and a permutation index is introduced:

```gams
set j / j1*j2 /, k / k1*k5 / 
jk(j,k) / j1.k3, j1.k5, j2.k1 / 
$eval pmax fact(card(jk)) 
p / p1*p%pmax% / 
pall(p,j,k,j,k); 
option pall > jk; 
option pall:0:0:1; 
display pall;
```

For multi-dimensional sets, the permutation operator follows the same logic. Consider the following example:
The display statement results in:

```plaintext
    8 SET pall
p1.j1.k3.j1.k3
p1.j1.k5.j1.k5
p1.j2.k1.j2.k1
p2.j1.k3.j1.k3
p2.j1.k5.j2.k1
p2.j2.k1.j1.k5
p3.j1.k3.j1.k5
p3.j1.k5.j1.k3
p3.j2.k1.j2.k1
p4.j1.k3.j1.k5
p4.j1.k5.j2.k1
p4.j2.k1.j1.k3
p5.j1.k3.j2.k1
p5.j1.k5.j1.k3
p5.j2.k1.j1.k3
p6.j1.k5.j1.k5
p6.j1.k3.j2.k1
p6.j2.k1.j1.k3
```

In addition to set elements, it is also possible to permute numerical data in a GAMS parameter. Consider the following example where all permutations of the numerical values stored in the one-dimensional parameter `a(i)` are computed:

```plaintext
set i / i1*i3 /
$eval pmax fact(card(i))
p / p1*p%pmax% /;
Parameter a(i) /i1 1, i2 2, i3 3/
    pall_a(p,i);
option pall_a > a;
option pall_a:0:1:1;
display pall_a;
```

The display statement results in:

```plaintext
    8 PARAMETER pall_a
      i1   i2   i3
   p1    1    2    3
   p2    1    3    2
   p3    2    1    3
   p4    2    3    1
   p5    3    1    2
   p6    3    2    1
```

The permutation can also be extended to multi-dimensional parameters. Consider the following example where all permutations of the numerical values stored in the two-dimensional parameter `b(j,k)` are computed:
4.35 System Attributes

4.35.1 Introduction

System attributes give access to string constants in the GAMS system environment. We will refer to these string constants as system suffixes. Moreover, there are attributes that can be used as data elements for user symbols. This way one can get access e.g. to the solvers (as set elements) in the system. We will refer to these data elements as system data. Two special system data attributes, namely powerSetLeft and powerSetRight, for a three dimensional set setName(n,s,b) create data which can be interpreted as a numbering system with base b and s digits. Finally, a system attribute can provide access to a system function that was hidden by the compiler because of a name conflict with a user symbol.

4.35.2 System Suffixes

System suffixes contain information about the GAMS system environment during a run. There are two ways to reference them: system.suffix and %system.suffix%. Here system is a keyword and .suffix is the name of the system suffix. A full list is given in section List of all System Suffixes below. Note that system.suffix references the execution-time version of the system suffix and %system.suffix% references the compile-time version resulting in a textual replacement. The execution-time system suffix can only be used in the context of put files. Consider the following example where both versions are used:

```
file fx;
put fx 'lp:' system(LP);
option lp=gurobi;
put / 'lp:' system(LP)
put / 'lp:%system(LP)%';
```

Observe that the compile-time string %system(LP)% is evaluated at compile time and does not change. However, the execution-time system suffix system(LP) is evaluated at execution time and it does change. It is first CPLEX, since CPLEX is the default LP solver. Then it changes to GUROBI as a result of the execution-time command option LP=gurobi;. The resulting put file fx.put follows:
4.35.2.1 Execution-Time System Suffixes

Execution-time system suffixes are only used in the context of the put writing facility with commands like `put`, `putclose` and `put utility`. Consider the following simple example. We have the files `x.gms` and `dummy.gms`, where `x.gms` contains just the following line:

```gams
file fx; put fx system.version;
```

The file `dummy.gms` contains the following:

```gams
$exit;
```

We run these files with the following calls:

```bash
> c:\gams\win64\24.7\gams x.gms action=c s=putVersion
```

Note that the value `c` for the command-line parameter `action` causes the first input file to be compiled only. The result is saved in the work file `putVersion`. Then we restart and execute `dummy.gms`. Now, depending on the GAMS release version, the `put` file `fx` will have a different content:

- `c:\gams\win64\24.7\gams dummy.gms r=putVersion` // this produces a put file with content "GAMS Rev 247"
- `c:\gams\win64\24.8\gams dummy.gms r=putVersion` // this produces a put file with content "GAMS Rev 248"

In contrast, the value of the compile-time system suffix `%system.version%` is determined at compile time. Suppose we change the file `x.gms` to contain the following line:

```gams
file fx; put fx "%system.version%";
```

The file `dummy.gms` is not changed and we use the same calls as above. Note that the `put` file will have the same content for both versions of GAMS:

- `c:\gams\win64\24.7\gams dummy.gms r=putVersion` // this produces a put file with content "GAMS Rev 247"
- `c:\gams\win64\24.8\gams dummy.gms r=putVersion` // this produces a put file with content "GAMS Rev 248"

4.35.2.2 Compile-Time System Suffixes

Compile-time system suffixes may be placed anywhere a compile-time variables makes sense, including code for conditional compilation. The quoting of these compile-time system suffixes depends on their use in the code. Consider the following example:

```gams
$set systemDATE %system.date%
display "system.date", "%system.date%";
$ifi %system.LP% == Xpress $log "LP solver is Xpress."
```

The resulting output generated by the display statement follows:

```
---- 3 system.DATE
11/07/16
```

4.35.2.3 List of all System Suffixes

In the following table all system suffixes are listed.
<table>
<thead>
<tr>
<th>System Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildCode</td>
<td>System build code</td>
</tr>
<tr>
<td>CNS</td>
<td>CNS solver that is active for <strong>CNS model type</strong></td>
</tr>
<tr>
<td>ComputerName</td>
<td>Operating system computer name</td>
</tr>
<tr>
<td>Date</td>
<td>Job date</td>
</tr>
<tr>
<td>DateFl</td>
<td>Job date format</td>
</tr>
<tr>
<td>DirSep</td>
<td>File or directory separator in file names</td>
</tr>
<tr>
<td>DNLP</td>
<td>DNLP solver that is active for <strong>DNLP model type</strong></td>
</tr>
<tr>
<td>Elapsed</td>
<td>Elapsed time in seconds since start of job</td>
</tr>
<tr>
<td>EMP</td>
<td>EMP solver that is active for <strong>EMP model type</strong></td>
</tr>
<tr>
<td>Error</td>
<td>Used to indicate unknown suffix</td>
</tr>
<tr>
<td>ErrorLevel</td>
<td>System Errorlevel</td>
</tr>
<tr>
<td>FE</td>
<td>File extension</td>
</tr>
<tr>
<td>FileSys</td>
<td>Name of the operating system (<strong>MSNT</strong> for Windows, <strong>UNIX</strong> for Linux and MacOS)</td>
</tr>
<tr>
<td>FN</td>
<td>File name</td>
</tr>
<tr>
<td>FP</td>
<td>File path</td>
</tr>
<tr>
<td>GamsRelease</td>
<td>GAMS Release number</td>
</tr>
<tr>
<td>GamsReleaseMaint</td>
<td>GAMS Release number with maintenance number suffix</td>
</tr>
<tr>
<td>GamsVersion</td>
<td>GAMS version number</td>
</tr>
<tr>
<td>GdxFileNameIn</td>
<td>GDX file name for input, set by <em>put_utility</em> command <strong>gdxin</strong></td>
</tr>
<tr>
<td>GdxFileNameOut</td>
<td>GDX file name for output, set by <em>put_utility</em> command <strong>gdxout</strong></td>
</tr>
<tr>
<td>GString</td>
<td>GAMS system audit string (exact GAMS version being used)</td>
</tr>
<tr>
<td>HostPlatform</td>
<td>Host platform</td>
</tr>
<tr>
<td>IFile</td>
<td>Input file</td>
</tr>
<tr>
<td>ILine</td>
<td>Current source line number being executed</td>
</tr>
<tr>
<td>IncLine</td>
<td>Include file line</td>
</tr>
<tr>
<td>IncName</td>
<td>Include file name</td>
</tr>
<tr>
<td>IncParent</td>
<td>Include file parent</td>
</tr>
<tr>
<td>IncParentL</td>
<td>Include file parent line number</td>
</tr>
<tr>
<td>IsAlphaBeta</td>
<td>Indicates an Alpha or Beta build</td>
</tr>
<tr>
<td>JobHandle</td>
<td>Job handle of last async call</td>
</tr>
<tr>
<td>Lice1</td>
<td>License display line 1</td>
</tr>
<tr>
<td>Lice2</td>
<td>License display line 2</td>
</tr>
<tr>
<td>LicenseDateEvalN</td>
<td>Evaluation date (numeric)</td>
</tr>
<tr>
<td>LicenseDateEvalS</td>
<td>Evaluation date (string)</td>
</tr>
<tr>
<td>LicenseDateMaintN</td>
<td>Maintenance date (numeric)</td>
</tr>
<tr>
<td>LicenseDateMaintS</td>
<td>Maintenance date (string)</td>
</tr>
<tr>
<td>LicenseDateN</td>
<td>License date (numeric)</td>
</tr>
<tr>
<td>LicenseDateS</td>
<td>License date (string)</td>
</tr>
<tr>
<td>LicenseDateSysN</td>
<td>GAMS module system date (numeric)</td>
</tr>
<tr>
<td>LicenseDateSysS</td>
<td>GAMS module system date (string)</td>
</tr>
<tr>
<td>LicenseDaysEval</td>
<td>License evaluation days left</td>
</tr>
<tr>
<td>LicenseDaysMaint</td>
<td>License maintenance days left</td>
</tr>
<tr>
<td>LicenseDC</td>
<td>License number</td>
</tr>
<tr>
<td>LicenseEval</td>
<td>License is eval license</td>
</tr>
<tr>
<td>LicenseFileName</td>
<td>The file name of the license file currently used</td>
</tr>
<tr>
<td>LicenseID</td>
<td>License ID string</td>
</tr>
<tr>
<td>System Suffix</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LicenseInstitution</td>
<td>License holding institution</td>
</tr>
<tr>
<td>LicenseLevel</td>
<td>GAMS license level</td>
</tr>
<tr>
<td>LicenseLevelText</td>
<td>GAMS license level text</td>
</tr>
<tr>
<td>LicenseLicensee</td>
<td>License Holder</td>
</tr>
<tr>
<td>LicenseMudText</td>
<td>License MUD type</td>
</tr>
<tr>
<td>LicensePlatform</td>
<td>License Platform abbreviation</td>
</tr>
<tr>
<td>LicensePlatformText</td>
<td>License Platform Text</td>
</tr>
<tr>
<td>LicenseStatus</td>
<td>License validation status code</td>
</tr>
<tr>
<td>LicenseStatusText</td>
<td>License validation status text</td>
</tr>
<tr>
<td>LicenseType</td>
<td>License type</td>
</tr>
<tr>
<td>LicenseVendor</td>
<td>License Vendor</td>
</tr>
<tr>
<td>Line</td>
<td>Line number in source code</td>
</tr>
<tr>
<td>ListLine</td>
<td>Line number on listing file</td>
</tr>
<tr>
<td>LP</td>
<td>LP solver that is active for LP model type</td>
</tr>
<tr>
<td>MACAddress</td>
<td>MAC address of the first network adapter</td>
</tr>
<tr>
<td>MaxInput</td>
<td>Max input line length that can be processed</td>
</tr>
<tr>
<td>MCP</td>
<td>MCP solver that is active for MCP model type</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory (in Mb) in use</td>
</tr>
<tr>
<td>MINLP</td>
<td>MINLP solver that is active for MINLP model type</td>
</tr>
<tr>
<td>MIP</td>
<td>MIP solver that is active for MIP model type</td>
</tr>
<tr>
<td>MIQCP</td>
<td>MIQCP solver that is active for MIQCP model type</td>
</tr>
<tr>
<td>MPEC</td>
<td>MPEC solver that is active for MPEC model type</td>
</tr>
<tr>
<td>NLP</td>
<td>NLP solver that is active for NLP model type</td>
</tr>
<tr>
<td>NullFile</td>
<td>The null filename</td>
</tr>
<tr>
<td>OFile</td>
<td>Output (Listing) file</td>
</tr>
<tr>
<td>OPage</td>
<td>Current page number in output (listing) file</td>
</tr>
<tr>
<td>Page</td>
<td>Current page number</td>
</tr>
<tr>
<td>PFile</td>
<td>Current put file</td>
</tr>
<tr>
<td>Platform</td>
<td>Job platform (DEX for MacOS, LEX for Linux, WEX for Windows)</td>
</tr>
<tr>
<td>PrLine</td>
<td>Line on listing page</td>
</tr>
<tr>
<td>Procname</td>
<td>Model types LP MIP etc</td>
</tr>
<tr>
<td>PrPage</td>
<td>Listing page number</td>
</tr>
<tr>
<td>PutFileName</td>
<td>The filename of the currently active PUT file</td>
</tr>
<tr>
<td>QCP</td>
<td>QCP solver that is active for QCP model type</td>
</tr>
<tr>
<td>RDate</td>
<td>Restart file date</td>
</tr>
<tr>
<td>ReDirLog</td>
<td>Append redirection string into the logfile</td>
</tr>
<tr>
<td>RFile</td>
<td>Restart file name</td>
</tr>
<tr>
<td>RMINLP</td>
<td>RMINLP solver that is active for RMINLP model type</td>
</tr>
<tr>
<td>RMIP</td>
<td>RMIP solver that is active for RMIP model type</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>RMIQCP solver that is active for RMIQCP model type</td>
</tr>
<tr>
<td>RMPEC</td>
<td>RMPEC solver that is active for RMPEC model type</td>
</tr>
<tr>
<td>RTime</td>
<td>Restart file time</td>
</tr>
<tr>
<td>SFile</td>
<td>Save file name</td>
</tr>
<tr>
<td>SString</td>
<td>Subsystem (Solver) audit (last solver used)</td>
</tr>
<tr>
<td>Tab</td>
<td>Tab character</td>
</tr>
<tr>
<td>TClose</td>
<td>Time to save GAMS</td>
</tr>
<tr>
<td>TComp</td>
<td>Time to compile</td>
</tr>
<tr>
<td>TExec</td>
<td>Time to execute</td>
</tr>
</tbody>
</table>
### 4.35 System Attributes

<table>
<thead>
<tr>
<th>System Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Job time</td>
</tr>
<tr>
<td>Title</td>
<td>Current listing title</td>
</tr>
<tr>
<td>TStart</td>
<td>Time to restart GAMS</td>
</tr>
<tr>
<td>UserConfigDir</td>
<td>User writable directory that is searched for gamsconfig.yaml</td>
</tr>
<tr>
<td>UserDataDir</td>
<td>User writable directory that is searched for gamslice.txt and others</td>
</tr>
<tr>
<td>UserName</td>
<td>Operating system user name</td>
</tr>
<tr>
<td>VerID</td>
<td>GAMS version ID</td>
</tr>
<tr>
<td>Version</td>
<td>GAMS compiler version</td>
</tr>
</tbody>
</table>

A model that prints all system suffixes with its current values to a put file can be found in model [SSUFFIX].

#### 4.35.3 System Data

Some of the system relevant information does not fit into a single string. Such system data can be stored in GAMS symbols. The system data for this symbol can be accessed in the data statement of the symbol, e.g. `set allSolvers / system.solverNames /`. Now you can work with the solver names as you can work with any set. The system data can be viewed as an internal set, so it can be you can to create more complex GAMS symbols with this information. For example:

```gams
set seq / 1*1000 /;
set solvermap(seq,*) / set.seq:system.solverNames /;
parameter solverpar(*) / system.solverNames 1 /;
```

Here is the list and dimensionality of the system data:

<table>
<thead>
<tr>
<th>System Data</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dollarOptions</td>
<td>1</td>
<td>Dollar control options</td>
</tr>
<tr>
<td>empty</td>
<td>1</td>
<td>Empty label</td>
</tr>
<tr>
<td>gamsFunctions</td>
<td>1</td>
<td>Intrinsic functions</td>
</tr>
<tr>
<td>gamsParameters</td>
<td>1</td>
<td>Command line parameters</td>
</tr>
<tr>
<td>gamsParameterSynonymMap</td>
<td>2</td>
<td>Map between command line parameters and their synonyms</td>
</tr>
<tr>
<td>gamsParameterSynonyms</td>
<td>1</td>
<td>Synonyms for command line parameters</td>
</tr>
<tr>
<td>GUSSModelAttributes</td>
<td>1</td>
<td>Model attributes stored for GUSS scenarios</td>
</tr>
<tr>
<td>modelTypes</td>
<td>1</td>
<td>Model types, e.g. LP, MIP, ...</td>
</tr>
<tr>
<td>platforms</td>
<td>1</td>
<td>Platform code</td>
</tr>
<tr>
<td>powerSetLeft</td>
<td>3</td>
<td>Numbering system with base b and s digits</td>
</tr>
<tr>
<td>powerSetRight</td>
<td>3</td>
<td>Numbering system with base b and s digits</td>
</tr>
<tr>
<td>predefinedSymbols</td>
<td>1</td>
<td>Predefined symbols, e.g. SameAs, Diag, ...</td>
</tr>
<tr>
<td>setConstants</td>
<td>1</td>
<td>System data names (this list)</td>
</tr>
<tr>
<td>solverNames</td>
<td>1</td>
<td>Names of solvers and tools</td>
</tr>
<tr>
<td>solverPlatformMap</td>
<td>2</td>
<td>Map between solvers and platforms</td>
</tr>
<tr>
<td>solverTypePlatformMap</td>
<td>3</td>
<td>Map between solvers, model types and platforms</td>
</tr>
<tr>
<td>systemSuffixes</td>
<td>1</td>
<td>System suffixes</td>
</tr>
</tbody>
</table>
Here is a secondary list of system attributes provide access to system data. This information allows to print the license memorandum [LICEMEMO] and is only used for internal purposes:

<table>
<thead>
<tr>
<th>System Data</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>clipCodes</td>
<td>1</td>
</tr>
<tr>
<td>components</td>
<td>1</td>
</tr>
<tr>
<td>clipComponentMap</td>
<td>2</td>
</tr>
<tr>
<td>componentSolverMap</td>
<td>2</td>
</tr>
<tr>
<td>gamsLicenseTypes</td>
<td>1</td>
</tr>
<tr>
<td>gamsLicenses</td>
<td>1</td>
</tr>
<tr>
<td>vendors</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.35.3.1 Power Set

The very special system attribute `powerSetLeft` and `powerSetRight` do not provide access to static data relevant to the system but produce data based on existing sets `b` and `s`. This system data produced can be interpreted as a numbering system with base `b` and `s` digits. The total number of tuples created is `power(card(b), card(s))`. The first index of the resulting set has to have at least this size. The following small example demonstrates the functionality of `powerSetRight`:

```plaintext
$set digits 3
set s / s1*s%digits%/ , b / 0,1 /;
$eval nMax power(card(b),card(s))
set n / n1*n%nMax%/ ;
x(n,s,b) / system.powerSetRight /;
option x:0:0:%digits%; display x;
```

With 3 digits the display results in the following output:

```
---- 6 SET x
 n1.s1.0,  n1.s2.0,  n1.s3.0
 n2.s1.0,  n2.s2.0,  n2.s3.1
 n3.s1.0,  n3.s2.1,  n3.s3.0
 n4.s1.0,  n4.s2.1,  n4.s3.1
 n5.s1.1,  n5.s2.0,  n5.s3.0
 n6.s1.1,  n6.s2.0,  n6.s3.1
 n7.s1.1,  n7.s2.1,  n7.s3.0
 n8.s1.1,  n8.s2.1,  n8.s3.1
```

Changing `powerSetRight` to `powerSetLeft` reassigns the order of the digits. So the lowest digit is on the left:

```
---- 6 SET x
 n1.s1.0,  n1.s2.0,  n1.s3.0
 n2.s1.1,  n2.s2.0,  n2.s3.0
 n3.s1.0,  n3.s2.1,  n3.s3.0
 n4.s1.1,  n4.s2.1,  n4.s3.0
 n5.s1.0,  n5.s2.0,  n5.s3.1
 n6.s1.1,  n6.s2.0,  n6.s3.1
 n7.s1.0,  n7.s2.1,  n7.s3.1
 n8.s1.1,  n8.s2.1,  n8.s3.1
```
4.35.4 Access to Hidden Functions

Intrinsic functions are not keywords but they are stored in a list of predefined names. If a user program uses such a predefined name of a function for a user symbol, GAMS will hide the original function. For example,

```gams
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = uniform(0,1);
```

will result in a compilation error in the second line because GAMS expects `uniform` to be a set. The function `uniform` is hidden for this GAMS program. Renaming the user symbol that hides a function is often the best solution for this, but there are a few cases (including GAMS code already compiled and stored in a restart file) where renaming is not an option. A hidden function can be accessed via the system attribute `system.functionName`. So the following will successfully compile and execute:

```gams
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = system.uniform(0,1);
```

4.36 The Grid and Multi-Threading Solve Facility

4.36.1 Introduction

The GAMS Grid facility allows to take advantage of High Performance Computing Grids and systems with multiple CPUs. This language feature facilitates the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture.

A typical application uses a coarse grain approach involving hundreds or thousands of model solutions tasks which can be carried out in parallel. Examples include but are not limited to scenario analysis, Monte Carlo simulations, Lagrangian relaxation, decomposition algorithms and advanced solution approaches.

The grid features work on all GAMS platforms and have been tailored to many different environments, like the Condor Resource Manager, a system for high throughput computing from the University of Wisconsin-Madison. Researchers using Condor reported a delivery of 5000 CPU hours in 20 hours wall clock time.

Similarly, the GAMS Multi-Threading Solve Facility allows the asynchronous submission and collection of model solution tasks on a single, multi-threaded machine while using efficient in-memory communication between GAMS and the solver.

**Disclaimer:** The use of the term grid computing may be offensive to some purists in the computer science world. We use it very loosely to refer to a collection of computing components that allow us to provide high throughput to certain applications. One may also think of it as a resurrection of the commercial service bureau concept of some 30 years ago.

**Caution:** Although these features have been tested on all platforms and are part of our standard release we may change the approach and introduce alternative mechanisms in the future.

**Acknowledgments:** Prof. Monique Guignard-Spielberg and Antoine Sauré at that time at Wharton School at the University of Pennsylvania introduced us to parallel Lagrangian relaxation on the SUN Grid Environment. Prof. Michael Ferris from the University of Wisconsin-Madison adopted our original GAMS grid approach to the high throughput system Condor and helped to make this approach a practical proposition.
4.36.2 The Grid Facility: Basic Concepts

The Grid facility separates the solution process into several steps which then can be controlled separately. First we will review the steps taken during synchronous solution and then we will introduce the asynchronous or parallel solution steps.

When GAMS encounters a solve statement during execution it proceeds in three basic steps:

1. **Generation**: The symbolic equations of the model are used to instantiate the model using the current state of the GAMS data base. This instance contains all information and services needed by a solution method to attempt a solution. This representation is independent of the solver and computing platform.

2. **Solution**: The model instance is handed over to a solver and GAMS will wait until it terminates.

3. **Update**: The detailed solution and statistics are passed to GAMS from the solver to update the GAMS data base.

In most cases, the time taken to generate the model and update the data base with the solution will be much smaller than the actual time spent in a specific solver. The model generation will take a few seconds, whereas the time to obtain an optimal solution may take a few minutes to several hours or even longer. If sequential model solutions do not depend on each other, we can solve in parallel and update the data base in a random order. All we need is a facility to generate models, submit them for solution and continue. At a convenient point in our GAMS program we will then look for the completed solutions and update the data base accordingly. To summarize, solving in parallel entails two steps:

1. **Submission Loop**: In this phase we will generate and submit models for solutions that can be solved independently.

2. **Collection Loop**: The solutions of the previously submitted models are collected as soon as a solution is available. It may be necessary to wait for some solutions to complete by pausing the execution for some time.

Note that we have assumed that there will be no errors in any of these steps. Of course, this will not always be the case and elaborate mechanisms are in place to make the operation fail-safe.

**Note**

For scenario analysis the solver GUSS might be of particular interest. The model [GUSSGRID] demonstrates how GUSS is used together with the Grid facility.

4.36.3 The Grid Facility: A First Example

In this section we will illustrate the use of the basic grid facility with the model [QMEANVAR]. This model traces an efficiency frontier for restructuring an investment portfolio. Each point on the frontier requires the solution of independent quadratic mixed integer models. The original solution loop is shown below:

```gams
loop(p(pp),
   ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
   solve minvar min var using miqcp ;
   xres(i,p) = x.l(i);
   report(p,i,'inc') = xi.l(i);
   report(p,i,'dec') = xd.l(i);
);
```

This loop will save the solutions of the model minvar for different returns ret. As the solutions do not depend on the order in which they are carried out, we can rewrite this loop to operate in parallel.
4.36.3.1 The Submission Loop

The first step for solving in parallel using the Grid facility is to write the submission loop:

Parameter h(pp) 'model handles';
minvar.solveLink = 3;
loop(p(pp),
   ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
   solve minvar min var using miqcp;
   h(pp) = minvar.handle;
);

The model attribute solveLink controls the behavior of the solve statement. The value of 3 (which is equivalent to the compile-time constant %solveLink.Async Grid%) directs GAMS to generate the model and submit it for solution and then continue without waiting for the completion of the solution step. Thus with setting minvar.solveLink to 3 we activate grid computing.

A handle in the grid environment identifies the particular model and data instances available. The model attribute handle contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The specific numerical values of handles are assigned by GAMS and may be used to recover solutions and manage models that are solved on the grid. In our example, the handle parameter is h and the set of all model instances is pp. The handle values that are stored in h are later used to collect the solutions once the solution processes are completed.

4.36.3.2 The Collection Loop

We collect the solutions with the following collection loop:

loop(pp$handleCollect(h(pp)),
   xres(i,pp) = x.l(i);
   report(pp,i,'inc') = xi.l(i);
   report(pp,i,'dec') = xd.l(i);
);

Note that the dollar condition restricts the looping set pp to those elements which return a nonzero value to the function handleCollect(h(pp)). The function handleCollect tests the solution status for each element of the set pp. And if the solution is available, it is read into the GAMS data base. In this case the function returns a value of 1. If the solution is not ready to be retrieved, the value zero will be returned.

Observe that the collection loop above has one big flaw. If a solution has not been ready (that is if handleCollect equaled zero), it will not be retrieved. We need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. We will use a repeat until construct and the handle parameter h to control the loop to look only for the solutions that have not been loaded yet. The code follows:

repeat
   loop(pp$handleCollect(h(pp)),
      xres(i,pp) = x.l(i);
      report(pp,i,'inc') = xi.l(i);
      report(pp,i,'dec') = xd.l(i);
   );
   display$handleDelete(h(pp)) 'trouble deleting handles' ;
   h(pp) = 0;
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
Once we have extracted a solution we will set the handle parameter \( h \) to zero. In addition, we want to remove the instance from the system by calling the function \( \text{handleDelete} \) which returns zero if successful. No harm is done if it fails but we want to be notified via the conditional display statement. Before running the collection loop again, we may want to wait a while to give the system time to complete more solution steps. This is done with the function \( \text{readyCollect} \) which waits until another model instance with a handle in \( h \) is ready to be collected (or the optionally defined number of seconds has passed). The final wrinkle is to terminate if all model instances have been deleted from the system since their solutions were retrieved or after 100 seconds have elapsed, even if we did not get all solutions. This is accomplished with the function \( \text{timeElapsed} \) and is important, because if one of the solution steps fails our program would never terminate. Recall that the handle parameter \( h \) equals zero for all elements of the set \( pp \) whose related models have been solved and their solutions have been extracted. The last statement in the code above sets the results of the missed solves to \( \text{na} \) to signal the failed solve. The parameter \( h \) will now contain the handles of the failed solves for later analysis.

Alternatively, we could use the function \( \text{handleStatus} \) and collect the solutions that are stored in a GDX file. For example, we could write:

```gams
loop(pp$(handleStatus(h(pp)) = 2),
   minvar.handle = h(pp);
   execute_loadhandle minvar;
   xres(i,pp) = x.l(i);
   report(pp,i,'inc') = xi.l(i);
   report(pp,i,'dec') = xd.l(i);
);
```

The function \( \text{handleStatus} \) tests the solution process and returns the value 2 if the solution process has been completed and the results can be retrieved. The solution is stored in a GDX file which can be loaded in a way similar to other GDX solution points. First, we need to specify which solution to retrieve by setting the the model attribute \( \text{minvar.handle} \) to the appropriate value. Then we can use the statement \( \text{execute_loadhandle} \) to load the solution for the model \( \text{minvar} \) back into the GAMS data base.

**Note**

Except for the requirement of a model with a previously specified handle, the command \( \text{execute_loadhandle} \) operates like the procedure \( \text{execute_loadpoint} \).

Using the function \( \text{handleStatus} \) and the command \( \text{execute_loadhandle} \) instead of the simpler \( \text{handleCollect} \), adds one more layer of control to the final collection loop. Now we need one additional \( \text{if} \) statement inside the collection loop above:

```gams
repeat
   loop(pp$h(pp),
      if(handleStatus(h(pp)) = 2,
         minvar.handle = h(pp);
         execute_loadhandle minvar;
         xres(i,pp) = x.l(i);
         report(pp,i,'inc') = xi.l(i);
         report(pp,i,'dec') = xd.l(i);
         display$h(pp)$handleDelete(h(pp)) 'trouble deleting handles' ;
         h(pp) = 0;
      );
   ) ;
   display$readyCollect(h, 100) 'Problem waiting for next instance to complete';
until card(h) = 0 or timeelapsed > 100;
```

Finally, we are ready to run the modified model.
4.36.3.3 The Execution Log

The execution log will contain some new information that may be useful for more advanced applications:

```plaintext
--- LOOPS pp = p1
--- 46 rows 37 columns 119 non-zeroes
--- 311 nl-code 7 nl-non-zeroes
--- 14 discrete-columns
--- Submitting model minvar with handle grid137000002
--- Executing after solve
... 
--- GDXin=C:\answerv5\gams_srcdev\225j\grid137000003\gmsgrid.gdx
--- Removing handle grid137000003
```

Note that the log contains some additional information about the submission, retrieval and removal of the solution instance. In the following sections we will make use of this additional information.

The execution log does not contain any solver log. Though, if a logOption value is set, that triggers writing to a file (i.e. 2 or 4), the solver log of each job will be written to the corresponding folder in the Grid Directory. So, if the solver log should be inspected when the job is done, one needs to make sure, that these folders do not get cleaned automatically, e.g. by setting the gridDir command line parameter.

For a complete example for grid computing, see the grid enabled transport model [TRNSGRID].

Observe that we have made no assumptions about what kind of solvers and what kind of computing environment we will operate. The example above is completely platform and solver independent and it runs on a Windows laptop or on a massive grid network like the Condor system without any changes in the GAMS source code.

4.36.4 Advanced Use of Grid Features

In this section we will describe a few special application requirements and show how this can be handled with the current system. Some of those applications may involve thousands of model instances with solution times of many hours each. Some may fail and require resubmission. More complex examples require communication and the use of GAMS facilities like the Branch-and-Cut-and-Heuristic Facility (BCH), which submit other models from within a running solver.

Imagine a situation with thousands of model instances each taking between minutes and many hours to solve. We will break the master program into a submitting program, an inquire program and a final collection program. We will again use the model [QMEANVAR] to demonstrate the principle. We will split the code of the modified [QMEANVAR] GAMS code into three components: qsubmit, qcheck and qreport.

4.36.4.1 Very Long Job Durations: The Submitting Program

The file qsubmit.gms will include everything up to and including the new submission loop. To save the instances we will need a unique grid directory gdir and to restart the problem we will have to create a save file. For details on the save and restart facility in GAMS, see chapter The Save and Restart Feature. When running the first job, we will use the command line parameter save or its synonym s to create the required save file:

```plaintext
> gams qsubmit s=submit gdir=c:\test\grid
```
4.36.4.2 Very Long Job Durations: The Inquire Program

The solution of all the model instances may take hours. From time to time we may run a quick inquiry job to learn about the status. The following program `qcheck.gms` will list the current status:

```gams
Parameter status(pp,*);
Scalar handle;
Acronym BadHandle, Waiting, Ready;
loop(pp,
  handle = handleStatus(h(pp));
  if(handle=0,
    handle = BadHandle;
  elseif handle=2,
    handle = Ready;
  minvar.handle = h(pp);
  execute_loadhandle minvar;
  status(pp,'solvestat') = minvar.solvestat;
  status(pp,'modelstat') = minvar.modelstat;
  status(pp,'seconds') = minvar.resusd;
  else
    handle = Waiting;
  );
  status(pp,'status') = handle;
); display status;
```

For details on the model attributes referenced in the code above, see `handle`, `solveStat`, `modelStat` and `resUsd`. To run the program above, we will restart from the previous save file by using the command line parameter `restart` or its synonym `r`.

```bash
> gams qcheck r=submit gdir=c:\test\grid
```

The output generated by the display statement may look like the following:

```
---- 173 PARAMETER status

<table>
<thead>
<tr>
<th>solvestat</th>
<th>modelstat</th>
<th>seconds</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1 1.000</td>
<td>1.000</td>
<td>0.328</td>
<td>Ready</td>
</tr>
<tr>
<td>p2 1.000</td>
<td>1.000</td>
<td>0.171</td>
<td>Ready</td>
</tr>
<tr>
<td>p3</td>
<td></td>
<td></td>
<td>Waiting</td>
</tr>
<tr>
<td>p4</td>
<td></td>
<td></td>
<td>Waiting</td>
</tr>
<tr>
<td>p5 1.000</td>
<td>1.000</td>
<td>0.046</td>
<td>Ready</td>
</tr>
</tbody>
</table>
```

We may want to do some more detailed analysis on one of the solved model instances. The respective program, called `qanalyze.gms`, may include the following lines of code:

```gams
$if not set instance $abort --instance is missing
if(not handleStatus(h('%instance%')),
  abort$yes 'model instance %instance% not ready');
minvar.handle = h('%instance%');
execute_loadhandle minvar;
display x.l,xi.l,xd.l;
...
```

For information on dollar control options, see chapter Dollar Control Options, especially the detailed descriptions of the options `$if` and `$abort`. Note that `instance` is a compile-time variable. The program may be called using a double dash parameter, which defines and sets a GAMS compile-time variable:

```bash
> gams qanalyze r=submit gdir=c:\test\grid --instance=p4
```
4.36.4.3 Very Long Job Durations: The Collection Program

Once all jobs are completed we are ready for the collection loop. For simplicity, we will not include the repeat loop, because we would not run the final collection program unless we were satisfied that we got most of the solutions that we wanted. The file `qreport.gms` could look like the following:

```plaintext
loop(pp$handleStatus(h(pp)),
  minvar.handle = h(pp);
  execute_loadhandle minvar;
  xres(i,pp) = x.l(i);  
  report(pp,i,'inc') = xi.l(i);
  report(pp,i,'dec') = xd.l(i);
  display$handleDelete(h(pp)) 'trouble deleting handles' ;
  h(pp) = 0;
);
  xres(i,pp)$h(pp) = na;
  ...
```

We will restart the program above from the `save file` that was created earlier:

```
> gams qreport r=submit gdir=c:\test\grid
```

Note that it is not necessary to run the job from the same directory that we have used for the initial submission; it is even possible to use a different operating system.

4.36.5 Summary of Grid Features

We introduced several GAMS features to facilitate the asynchronous or parallel execution of the solve statement. These GAMS features are summarized in the following subsections.

In addition to the features described below, the option or command line parameter `ThreadsAsync` was introduced. `ThreadsAsync` controls the number of threads or CPU cores that are used in multi-threading computing.

4.36.5.1 Grid Handle Functions

The grid handle functions are listed in Table 1. For details on functions in GAMS in general and complete lists of all GAMS functions, see section Functions. Note that the desired return values - the return values that indicate that no error has occurred - are marked with bold letters.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
</table>
| handleCollect(HANDLE) | Tests if the solve of the model instance identified by HANDLE is done: if so, it loads the solution into the GAMS data base. If the option asyncSolLst is active the solution listing is printed to the listing file. **Note** that handleCollect ignores the setting of the option SolveOpt and always uses the default value *merge*. | 0: The model instance was not ready or could not be loaded.  
>0: **The model instance solution has been loaded.** (When using the Grid facility, this is always 1; when using the multi-threading option, this returns the thread ID used.) |
| handleStatus(HANDLE) | Tests if the solve of the model instance identified by HANDLE is done.  
**Note** that there are compile-time constants that are related to this function. | 0: The model instance is not known to the system.  
1: The model instance exists but the solution process is incomplete.  
2: **The solution process has terminated and the solution is ready for retrieval.**  
3: The solution process signaled completion but the solution cannot be retrieved. |
| handleDelete(HANDLE) | Deletes the model instance identified by HANDLE and returns a numerical indicator of the status of the deletion. If the HANDLE given is not valid, an execution error is triggered. | 0: **The model instance has been removed.**  
1: The argument HANDLE is not a legal handle.  
2: The model instance is not known to the system.  
3: The deletion of the model instance encountered errors. |
| handleSubmit(HANDLE) | Resubmits the model instance identified by HANDLE for solution. In case of a nonzero return an execution error is triggered. | 0: **The model instance has been resubmitted for solution.**  
1: The argument HANDLE is not a legal handle.  
2: The model instance is not known to the system.  
3: The completion signal could not be removed.  
4: The resubmit procedure could not be found.  
5: The resubmit process could not be started. |
### 4.36 The Grid and Multi-Threaded Solve Facility

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>readyCollect(HANDLES[,maxWait])</td>
<td>Waits until a model solution is ready to be collected. HANDLES must be either a scalar or parameter containing one or more model handles or a model with its handle attribute. MaxWait specifies the maximum time to wait in seconds, the default value is +inf.</td>
<td>0: One or more of the requested jobs is/are ready. 1: There is no active job to wait for. 2: The handle symbol is empty. 3: The argument is not a legal handle. 4: User specified time-out (using a solveLink = 6 handle). 5: User specified time-out (using a solveLink = 3 handle). 8: Unknown error (should not happen).</td>
</tr>
</tbody>
</table>

**Table 1: Grid Handle Functions**

Note that GAMS might issue execution errors which could give additional information that may help to identify the source of problems. The function execError may be used to get and set the number of execution errors.

#### 4.36.5.2 Grid Model Attributes

Model attributes are introduced in section Model Attributes. The following three model attributes are particularly relevant for grid computing:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solveLink</td>
<td>Specifies the solver linking conventions. The following values direct the solve statement to use grid computing or multi-thread computing: 3, 4, 6 and 7. Note that the default for this model attribute can be set as command line parameter and option statement. For more information, see the detailed description.</td>
</tr>
<tr>
<td>handle</td>
<td>Specifies the current instance handle. This is used to identify a specific model instance and to provide additional information needed for the process signal management (compare subsections The Submission Loop and The Collection Loop above).</td>
</tr>
<tr>
<td>number</td>
<td>Specifies the current instance number. Any time a solve is attempted for a model, the instance number is incremented by one and the handle is update accordingly. The instance number can be reset by the user which then resynchronizes the handle.</td>
</tr>
</tbody>
</table>

**Table 2: Grid Handle Attributes**

#### 4.36.5.3 Grid Solution Retrieval

As an alternative to the function handleCollect a solution may be retrieved with the following statement:

```plaintext
eexecute_loadhandle mymodel;
```

This statement will update the GAMS data base with the status and solution for the current instance of `mymodel`. Note that the underlying mechanism is a GDX file. Except for the requirement of a model with a previously specified handle, this command operates like the procedure execute_loadpoint. If the option asyncSolLst is active the solution listing is printed to the listing file.
4.36.5.4 The Grid Directory

The instantiated (generated) models and their corresponding solutions are kept in unique directories that may be reached from the submitting system. Each GAMS job may have only one grid directory. By default, the grid directory is assumed to be the scratch directory. This may be overwritten by using the GAMS command line parameter GridDir, or short GDir. An example follows.

> gams myprogram ... GDir=gridpath

If gridpath is not a fully qualified name, the name will be completed using the current directory. If the grid path does not exist, an error will be issued and the GAMS job will be terminated. A related GAMS parameter is ScrDir or short SD.

Recall the following default mechanism: When a GAMS job is started a unique process directory is created in the current directory. These directories are named 225a to 225zz. When a GAMS job terminates, the system will remove the process directory at the completion of a GAMS job. Any file that has not been created by the GAMS core system will be flagged. If the call gamskeep instead of gams is used, another exit script will be activated that results in the process directory to be kept.

Note that if we do not specify a scratch directory, the scratch directory will be the same as the process directory. If we do not specify a grid directory, the grid directory will be the same as the scratch directory.

Observe that if we assume that some of the model instances may fail or we want to break the GAMS program into several pieces to run as separate jobs, we need to be careful not to remove the model instance we have not completely processed. In such cases we have to use the parameter GridDir, so that we may access previously created model instances.

4.36.6 The Grid Facility: Architecture and Customization

The current Grid facility relies on very basic operating system features and does not attempt to offer real and direct job or process control. The file system is used to signal the completion of a submitted task and GAMS has currently no other way to interact with the submitted process directly, like forcing termination or change the priority of a submitted task. This approach has its obvious advantages and disadvantages. There are a number of attempts to use grid computing to provide value added commercial remote computing services.

When GAMS executes a solve with the option solveLink set to 3 it will perform the following steps:

1. Create a subdirectory in the GridDir with the name gridnnn. Here nnn stands for the numeric value of the handle. The handle value is the internal symbol ID number x 1e6 + the model instance number. For example, in the [QMEANVAR] example the first grid subdirectory was grid137000002.

2. Remove the completion signal in case the file already exists. Currently the signal is a file called finished. For example, grid137000002/finished.

3. Create or replace a GDX file called gmsgrid.gdx which will contain a dummy solution with failed model and solver status. This file will be overwritten by the final step of the solution process and will be read when calling execute_loadhandle.

4. Place all standard GAMS solver interface files into the above instance directory.

5. Execute the submission wrapper called gmsgrid.cmd under Windows or gmsgrid.run under Unix. These submission scripts are usually located in the GAMS system directory, they may be located via the current path if they are not found in the GAMS system directory.
The grid submission script `gmsgrid.cmd` or `gmsgrid.run` is called with four arguments that are needed to make a standard GAMS solver call: the solver executable file name, the solver control file name, the solver scratch directory, and the solver name. The submission script then does the final submission to the operating system. This final script will perform the following steps:

1. call the solver,
2. call a utility that will create the final GDX file `gmsgrid.gdx`,
3. set the completion signal `finished`.

If we want to use the function `handleSubmit` we will also have to create the script `gmsrerun.cmd` or `gmsrerun.run`. This script could later be used to resubmit the job.

For example, the default submission script for Windows is shown below:

```
@echo off
: gams grid submission script
:
: arg1 solver executable
: 2 control file
: 3 scratch directory
: 4 solver name
:
: gmscr_nx.exe processes the solution and produces 'gmsgrid.gdx'
:
: note: %3 will be the short name, this is needed because
: the START command cannot handle spaces or "...
: before we use %"3 will strip surrounding "...
: makes the name short
:
: gmsrerun.cmd will resubmit runit.cmd

echo @echo off > %3runit.cmd
echo %1 %2 %4 >> %3runit.cmd
echo gmscr_nx.exe %2 >> %3runit.cmd
echo echo OK ^> %3finished ^& exit >> %3runit.cmd

echo @start /b /belownormal %3runit.cmd ~> nul > %3gmsrerun.cmd

start /b /belownormal %3runit.cmd > nul

exit
```

### 4.36.6.1 Grid Submission Testing

The grid submission process can be tested on any GAMS program without having to change the source text. The option `solveLink=4` instructs the solve statement to use the grid submission process and then wait until the results are available. Note that the option `solveLink` may be set via a GAMS command line parameter, a GAMS option statement or via assignment to the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the `reslim` seconds allocated for this optimization job and report a failure if this limit has been exceeded. After successful or failed retrieval of the solution, GAMS will remove the grid directory, unless we have used the call `gamskeep` or have set the GAMS command line parameter `keep`.
4.36.7 Multi-Threading

As we have described in this chapter, each solve is handled in its own process space with the Grid facility. Recall that the Grid facility is activated by setting the option or model attribute `solveLink` to 3 or 4. If `solveLink` is set to 6 (or the compile-time constant `%solveLink.Async Threads%`) instead, a separate thread is used. This allows efficient in-memory communication between GAMS and the solver, like it is done if the option `solveLink` is set to 5 (or the compile-time constant `%solveLink.Load Library%`).

Apart from this, the multi-threading facility works in the same way as the Grid facility. The solve statement generates the model and passes it to the solver in a separate thread, then a handle of the model instance may be stored using the model attribute `handle` and the grid `handle` functions may be used to collect the solution and deal with the model instance, namely `handleCollect`, `handleDelete`, `handleStatus` and `readyCollect`.

Note that the option or command line parameter `ThreadsAsync` sets the maximum number of threads that should be used for the asynchronous solves.

The following matrix shows which solvers may be used with `solveLink = 6` on which platform:

<table>
<thead>
<tr>
<th>Solver</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONOPTD</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CONOPT4</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CONVERT</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>COPT</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CPLEX</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>GUROBI</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>HIGHS</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>IPOPT</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>MOSEK</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>OCTERACT</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>ODHCPLEX</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>OsiCplex</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>OsiGurobi</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SCIP</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SHOT</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SNOPLEX</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Soplex</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>XPRESS</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

If a solver is selected for which `solveLink = 6` is not supported on the corresponding platform, `solveLink = 3` will be used instead and it will be noted in the log.

4.36.7.1 Multi-threading Submission Testing

The multi-threading submission process may be tested on any GAMS program without having to change the source text. The option `solveLink = 7` (or, equivalently, the compile-time constant `%solveLink.Threads Simulate%`) instructs the solve statement to use the multi-threading submission process, wait until the results are available and then load the solution into the GAMS data base. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the `reslim` seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution GAMS will remove the thread handle.
4.37 Special Features for Mathematical Programs

4.37.1 Introduction

In this chapter we will introduce special GAMS features that are useful for specific model types. The features include model scaling, conic programming and features that facilitate mixed integer as well as indicator constraints, a feature that does not translate across solvers.

4.37.2 Special Mixed Integer Programming (MIP) Features

Some special features have been added to GAMS to help simplifying the modeling of Mixed Integer Programming (MIP) problems. In GAMS MIP is the model type for mixed integer linear programs, this section used MIP more generally, we consider model with discrete variables, including non-linear expressions and pure discrete problem. We will first present details on discrete variables in GAMS, then we will discuss how to customize priorities for the branching process. Next, we will cover the model attributes that are important for MIPs and we will conclude with some hints that will make mixed integer programming with GAMS easier.

4.37.2.1 Types of Discrete Variables

Variables and variable types are introduced in chapter Variables. GAMS provides six discrete variable types: binary, integer, sos1, sos2, semicont and semiint. In the following subsections we will present details and examples for each of these discrete variable types. Note that if any discrete variables feature in a model, it has to be a mixed integer model or one of the related model types, like MINLP or MIQCP. See section Classification of Models for a full listing of all GAMS model types.

4.37.2.1.1 Binary Variables

Binary variables can take values of 0 (zero) and 1 (one) only. They are declared as follows:

Binary Variable var_name [(index_list)] [text];

The keyword binary indicates that this is a binary variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as binary later. Consider the following code snippets from the orthogonal Latin Square model [LATIN]:

Sets k "rows" / row1*row4 /
    l "columns" / col1*col4 /
    v "values" / val1*val4 /;
alias (i,j,v);

Variables x(i,j,k,l) "pairs (i,j) allocated to cell(k,l)"
    z "some objective";
Binary Variable x;

Equations c1(i,j) "for each cell pick only one item pair"
    c1(i,j).. sum((k,l), x(i,j,k,l)) =e= 1;
Note that the binary variable $x$ is used in equation $c1$ to model the restriction that in each cell only one item pair is allowed. Binary variables are often used to model logical conditions such as imposing mutual exclusivity or complementarity.

Note that the default lower bound is 0 (zero) and the default upper bound is 1 (one). If the relaxed versions of the discrete models is solved, binary variables are treated like positive variables with the upper bound of 1. In addition, an infinite priority may be used to override binary specifications, see section Setting Priorities for Branching below for more information.

Even though the only possible values are 0 and 1, a solver might return a value for binary variable that is only close to 0 or 1. Every solver works with tolerances and also uses a tolerance to determine if a value is close enough to an integer values. So it is unwise to use code as $a(i)$(b.l(i)=1) = yes; because one will potentially miss some elements. A safe way to write such code is: $a(i)$(b.l(i)>0.5) = yes; Rounding the level of a binary variable after the solve is also possible, but it is not done by the solver or the solver link because even small rounding can lead to infeasibilities.

A binary variable can also have a truely fractional value after a solver if the model status does not indicate a feasible integer solution (model status 1 or 8).

### 4.37.2.1.2 Integer Variables

Integer variables are discrete variables that can take only values between their bounds. The user may change both bounds from the default value. The default lower bound is 0 (zero) and the default upper bound inside GAMS is $+\text{inf}$, and the same upper bound is passed on to the solver.

Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section Setting Priorities for Branching below for more information. Integer variables are declared as follows:

```
Integer Variable var_name [(index_list)] [text];
```

The keyword `integer` indicates that this is an integer variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as `integer` later. Consider the following code snippets from the power scheduling model [MAGIC]:

```
Sets
t "demand blocks" / 12pm-6am, 6am-9am, 9am-3pm, 3pm-6pm, 6pm-12pm /
g "generators" / type-1, type-2, type-3 /;

Variables
x(g,t) "generator output (1000mw)"
n(g,t) "number of generators in use"
cost "total operating cost (1)"
;

Integer Variable n;
```

The integer variable $n$ models the number of generators of various types that are in use at any of the time blocks.
4.37.2.1.3 Special Order Sets of Type 1 (SOS1)  
SOS1 variables are a set of variables, such that at most one variable within the group may have a nonzero value. This variable may take any positive value. Special ordered sets of type 1 are defined as follows:

SOS1 Variable var_name (index_list) [text];

The keyword SOS1 indicates that this is a SOS1 variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as SOS1 later. Consider the following example:

SOS1 Variable s1(i), t1(k,j), w1(i,j,k) ;

Note that the members of the innermost (the right-most) index belong to the same SOS set. For example in the sets defined above, s1 represents one special ordered set of type 1 with i elements, t1 defines k sets with j elements each and w1 defines (i,j) sets with k elements each.

The default bounds for SOS1 variables are zero and +inf. As with any other variable, the user may change these bounds. Further, the user may explicitly provide whatever convexity row that the problem may need through an equation that requires the members of the SOS1 set to be less than a certain value. Any such convexity row will implicitly define bounds on each of the variables.

Consider the following example:

SOS1 Variable s1(i); Equation defsoss1;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;

The equation defsoss1 implicitly defines the nonzero value that one of the elements of the SOS1 variable s1 may take as equal to or smaller than 3.5. Note that it is also possible that all variables s1 equal zero.

A special case arises when one of the elements of the set has to be nonzero and equal to a number, say 3.5. In this case equation defsoss1 will be:

defsoss1.. sum(i,s1(i)) =e= 3.5 ;

Frequently the nonzero value equals 1. As a result, the SOS1 variable is effectively a binary variable. It is only treated differently by the solver at the level of the branch and bound algorithm. For example, consider the following example where we want to model that one out of n options has to be selected. This is expressed as:

SOS1 Variable x(i); Equation defx;
defx.. sum(i, x(i)) =e= 1 ;

The variable x can be made binary without any change in meaning and the solution provided by the solver will be indistinguishable from the SOS1 case.

The use of special ordered sets may not always improve the performance of the branch and bound algorithm. If there is no natural order the use of binary variables may be a better choice. A good example of this is the classical assignment problem (see [H.P. Williams (2013) Model Building in Mathematical Programming], Wiley, Section 9.3.

Note that any model with SOS1 variables requires a MIP solver, because the solution process needs to impose the restrictions of at most one nonzero level values may be present.

For an example where SOS1 variables are used, see the production scheduling model  [PRODSCHX].
**4.37.2.1.4 Special Order Sets of Type 2 (SOS2)** SOS2 variables are a set of variables, such that at most two variables within the set may have nonzero values and these variables have to be adjacent. This requirement implies that the set is ordered, see chapter Sets as Sequences: Ordered Sets for details on ordered sets in GAMS. Note that the nonzero variables may take any positive value. Special ordered sets of type 2 are defined as follows:

```
SOS2 Variable var_name [(index_list)] [text];
```

The keyword **SOS2** indicates that this is a SOS2 variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as SOS2 later. Consider the following example:

```
Set i / i1*i5 /
SOS2 Variable s2(i), t2(k,j), w2(i,j,k);
```

The members of the innermost (the right-most) index belong to the same set. For example, in the sets defined above, s2 represents one special ordered set of type 2 with elements for each member of the set i. At most two variables s2 may be nonzero and they must reference adjacent elements of the set i. Note that the variables s2('i1') and s2('i2') are adjacent, but the variables s2('i1') and s2('i3') are not. Further, t2 defines k sets of SOS2 variables with j elements each and the adjacency requirement refers to the set j which must be ordered. Similarly, w2 defines (i,j) sets with k elements each and the adjacency requirement refers to the set k which must be ordered.

The default bounds for SOS2 variables are zero and +inf. As with any other variable, the user may change these bounds. SOS2 variables are most often used to model piece-wise linear approximations to nonlinear functions. The production scheduling model [PRODSCHX] shows SOS type formulations with binary, SOS1 and SOS2 sets.

Note that any model with SOS2 variables requires a MIP solver, because the solution process needs to impose the restrictions of adjacency and that no more than two nonzero level values may be present.

**4.37.2.1.5 Semi-Continuous Variables** Semi-continuous variables are either zero or above a given minimum level. This can be expressed algebraically as: either \( x = 0 \) or \( L \leq x \leq U \). By default, the lower bound \( L \) is 1 and the upper bound \( U \) is +inf. As usual, these bounds may be changed with the variable attributes .lo and .up. Semi-continuous variables are defined as follows:

```
SemiCont Variable var_name [(index_list)] [text];
```

The keyword **semicont** indicates that this is a semi-continuous variable and then the usual conventions for variable declarations” are followed. Alternatively, the variable may be declared first and specified as semicont later. Consider the following example:

```
SemiCont Variable x;
x.lo = 1.5; x.up = 23.1;
```

The slice of code above declares the variable x to be a semi-continuous variable that may either be zero or behave as a continuous variable between 1.5 and 23.1.

Note that any model with semi-continuous variables requires a MIP solver, because the solution process needs to impose the discontinuous jump between zero and the threshold value.
Note

- Not all MIP solvers allow semi-continuous variables. We recommend users to verify how the solver they are interested in handles semi-continuous variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-continuous variables.
- Semi-continuous variables are especially helpful if the upper bound is $+\infty$ and no implicit bound can be easily derived. If a finite upper bound is available it can be computational more efficient to replace the semi-continuous variable $sc$ with lower bound $sc_{\text{Low}}$ by a continuous variable $x$ and binary variable $b$ and the following equations:

  Equation \texttt{xForceLowerBnd} "Force x to be greater than scLow if b is 1"
  \texttt{xForceZero} "Force x to be zero if b is zero"

  \begin{verbatim}
  xForceLowerBnd.. x =g= scLow*b;
  xForceZero.. x =l= x.up*b;
  \end{verbatim}

4.37.2.1.6 Semi-integer Variables

Semi-integer variables are either zero or integer and above a given minimum value. This can be expressed algebraically as: either $x = 0$ or $x \in \{L, \ldots, U\}$. By default, the lower bound $L$ is 1 and the upper bound $U$ inside GAMS is $+\infty$ and the same values are passed on to the solver. As usual, these default bounds may be changed with the variable attributes .lo and .up. Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section Setting Priorities for Branching below for more information.

Semi-integer variables are defined as follows:

\texttt{SemiInt Variable var\_name [(index\_list)] [text];}

The keyword \texttt{semiint} indicates that this is a semi-integer variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as \texttt{semiint} later. Consider the following example:

\begin{verbatim}
SemiInt Variable x;
x.lo = 2; x.up = 25;
\end{verbatim}

The slice of code above declares the variable $x$ to be a semi-integer variable that may either be zero or take any integer value between 2 and 25. Note that the bounds for \texttt{semiint} variables have to take integer values, otherwise GAMS will flag an error during model generation. Note further, that any model with semi-integer variables requires a MIP solver.

Note

- Not all MIP solvers allow semi-integer variables. We recommend users to verify how the solver they are interested in handles semi-integer variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-integer variables.
• Semi-integer variables are especially helpful if the upper bound is +\( \text{inf} \) and no implicit bound can be easily derived (together with the appropriate IntVarUp setting). If a finite upper bound is available, it can be computationally more efficient to replace the semi-integer variable \( s_i \), with lower bound \( s_{i\text{Low}} \), by an integer variable \( i \) and a binary variable \( b \) and the following equations:

\[
\text{Equation iForceLowerBnd } \quad "\text{Force } i \text{ to be greater than } s_{i\text{Low}} \text{ if } b \text{ is 1}" \\
iForceZero \quad "\text{Force } i \text{ to be zero if } b \text{ is zero}";
\]

\[
iForceLowerBnd. \quad i \geq s_{i\text{Low}} \cdot b;
\]

\[
iForceZero. \quad i \leq i.\text{up} \cdot b;
\]

### 4.37.2.2 Setting Priorities for Branching

By setting priorities users may specify an order for choosing variables to branch on during a branch and bound search for MIP models. Without priorities the MIP algorithm will internally determine which variable is the most suitable to branch on. Priorities for individual variables may be used only if the model attribute .prioropt is set to 1; the respective GAMS statement is:

\[
mymodel.prioropt = 1;
\]

Here \( \text{mymodel} \) is the name of the model specified in the model statement. The default value is NA.

If the model attribute .prioropt is set to 1, the variable attribute .prior may be used to set the priorities of individual discrete variables. Note that there is one .prior value for each individual component of a multidimensional variable. Priorities may be set to any real value; the default value is 1. As a general rule, the most important variables should be given the highest priority. The highest priority is denoted by the lowest nonzero value in the .prior attribute. Functionally, the attribute .prior establishes in what order variables are to be branched on in the branch-and-bound algorithm while searching for a solution. Variables with a specific .prior value will be branched on earlier until all fractional variables with higher .prior values have been branched on.

Note

The variable attribute .prior of a discrete variable may be used to relax the discrete restriction on that variable: setting the .prior value to +\( \text{inf} \) will relax a variable permanently (or until .prior gets a finite value assigned). This relaxation is done independently of the model attribute .prioropt.

Consider the following example:

\[
z.\text{prior}(i, \text{\textquoteleft}small\textquoteleft) = 3;
z.\text{prior}(i, \text{\textquoteleft}medium\textquoteleft) = 2;
z.\text{prior}(i, \text{\textquoteleft}large\textquoteleft) = 1;
\]

In this example the variables \( z(i, \text{\textquoteleft}large\textquoteleft) \) are branched on before the variables \( z(i, \text{\textquoteleft}medium\textquoteleft) \), which in turn are branched on before the variables \( z(i, \text{\textquoteleft}small\textquoteleft) \).

Note that knowledge about the problem may help to determine which variables should be considered first. For example, consider a problem with a binary variable \( u \) representing a yes/no decision whether to build a factory and other binary variables representing equipment selections within that factory. We would naturally want to explore whether or not the factory should be built before considering what specific equipment to be purchased within the factory. Therefore we would set the priority values lower for \( u \). By assigning a higher priority - a lower value of the attribute .prior - to the build/nobuild decision variable \( u \), we can force this logic into the tree search and thus speed up computation time since uninteresting portions of the tree are left unexplored.
Note

- The lower the value given to the .prior suffix, the higher the priority for branching.
- All members of any SOS1 or SOS2 set should be given the same priority value since it is the set itself which is branched upon rather than the individual members of the set.
- While any value is accepted for .prior, many solvers scale all giving priorities in the integer range of 0,...,1000.
- Branching priorities were a very important feature in the early days of mixed integer programming. Nowadays, it is not easy to find branching priorities that improve on the solvers default selection.
- Global non-linear optimization solvers branch on continuous variables too (see, for example BARON). In GAMS one cannot set the branching priority of a continuous variable. Such branching priorities need to be communicated via a solver option file.

4.37.2.3 Miscellaneous Hints

We will conclude the discussion of mixed integer models in GAMS with this section where we offer a variety of hints that are meant to make special facilities of mixed integer programming solvers more accessible.

4.37.2.3.1 Model Attributes for Mixed Integer Programming in GAMS GAMS offers several model attributes that may be used to influence MIP solver performance or report on results of MIPs. These model attributes include Cheat, CutOff, NodLim, ObjEst, OptCA, OptCR, PriorOpt and TryInt.

4.37.2.3.2 The Branch and Cut and Heuristic Facility Hard MIP problems can be solved faster with the help of user supplied routines that generate cutting planes and good integer feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) automates all major steps necessary to define, execute and control the use of user defined routines within the framework of general purpose branch-and-cut codes. It is documented in Branch-and-Cut-and-Heuristic Facility (BCH).

4.37.2.3.3 Branch and Bound Output While the log output for each solver differs, some key figures are usually displayed for branch-and-bound based solvers. For example, solving a linear mixed integer model with CPLEX will yield output like the following:

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Left</td>
<td>Objective</td>
<td>IInf</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>0.0000</td>
<td>5.05646e+08</td>
</tr>
<tr>
<td>Found incumbent of value 0.000000 after 0.01 sec. (0.73 ticks)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>2.23031e+07</td>
<td>12</td>
</tr>
<tr>
<td>0 0</td>
<td>2.23031e+07</td>
<td>7</td>
</tr>
<tr>
<td>* 0+ 0</td>
<td>2.23031e+07</td>
<td>2.23031e+07</td>
</tr>
<tr>
<td>Found incumbent of value 2.2303094e+07 after 0.02 sec. (1.08 ticks)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixing integer variables, and solving final LP...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution satisfies tolerances.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIP Solution: 22303093.628684 (23 iterations, 0 nodes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Solve: 22303093.628684 (0 iterations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best possible: 22303113.950091</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absolute gap: 20.321407</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relative gap: 0.000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A brief explanation of the columns follows:

- **Node** is the number of branch and bound nodes so far.
- **Nodes Left** is the number of problems created during the branching process that are yet to be examined.
- **Objective** gives the current objective function value of the relaxes node problem.
- **IInf** gives the number of discrete variables with fractional solution levels.
- **Best Integer** gives the incumbent solution. Note the last solution in that column is not necessarily the global best solution.
- **Cuts/Best Bound** gives the current lower bound on the solution.
- **Cnt** gives the accumulated LP iteration count
- **Gap** gives the maximum percentage difference from the theoretical optimum.

Note that it is common that solves of mixed integer models end with a gap between the solution found and the best possible solution. This may be controlled by limits (e.g. time), solver options and model attributes like .OptCR and .OptCA.

### 4.37.2.3.4 Nonlinear MIPs

Modelers may wish to impose integer restrictions on nonlinear formulations combining two hard model types: MIP and NLP. Such MINLP models can be solved with a selection of solvers. Many solvers, e.g. DICOPT and SBB, provide a local optimum where others, e.g. ANTIGONE and BARON provide a global optimum. In most cases both types of MINLP solver make use of MIP and NLP solvers to calculate a solution. Such subsolvers need to be licensed for the solver to succeed.

### 4.37.2.3.5 Model Termination Conditions and Recommended Actions

Recall that the termination condition of the model after the solution process has been completed is stored in the model attribute .modelStat. A list of all possible model statuses is given in section Model Status. We can easily check for the existence of a feasible solution (status 1 and 8). Linear models and MINLP problems solved with global solvers can achieve the "OPTIMAL" status, given sufficient resources (time) and a setting of OptCR and OptCA to 0. All other cases do not yield a feasible integer solution. If a problem is reported as infeasible (status 4,5,10, and 19), it might be a good idea to see if the relaxed version of the model is already infeasible. Debugging models that are relaxed feasible but integer infeasible is very difficult.

### 4.37.2.3.6 Frequent Problems

There are some problems users frequently encounter either due to GAMS settings or problem characteristics:

#### Default bounds

One needs to be aware that while the GAMS upper bound for integer and semi-integer variables is +inf, the bound that is passed to the solver can be different, namely 100 (see the discussion about integer variables). This can lead to unexpected results (e.g. infeasibilities or suboptimal solutions declared as optimal).

#### Ending with a gap – large default for optCR (10%)

MIP solves often end with a gap between the solution found and the best possible solution. This is controlled by OptCR or OptCA or by non-convergence. Note that the default value of 0.1 for optCR is relatively large. Users may want to reduce this to a smaller value. We will discuss the other cause of a gap next.

#### The nonending quest

Integer programming is a quite desirable formulation technique. However, integer problems are theoretically hard and the solution process (in the worst case) of exponential complexity. There are many ways that focus on improving the solution time of the solver. As with all models scaling is important (especially when using bigM formulations). For particular problems many different formulations exist and the literature about a particular problem together with the ability of GAMS to rapidly prototype and experiment is the best constellation to get the best results for the problem at hand. For fine tuning, some MIP solvers provide automated tuning tools (see e.g. Cplex tuning) that tweak the solver options to get the best performance.
4.37.3 Model Scaling - The Scale Option

The rules for good scaling are exclusively based on algorithmic needs. GAMS has been developed to increase the efficiency of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as natural as possible. The units of measurement are one part of this natural notation. However, there is a potential conflict between what the modeler thinks is a good unit and what constitutes a well-scaled model.

4.37.3.1 The Scale Option

To facilitate the translation between a natural model and a well scaled model, GAMS has introduced the concept of a scale factor, both for variables and equations. The notations and definitions are quite simple. Scaling is turned off by default. Setting the model attribute .scaleopt to 1 turns on the scaling feature. For example,

```gams
model mymodel /all/ ;
mymodel.scaleopt = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The statement should be inserted somewhere after the model statement and before the solve statement. To turn scaling off again, mymodel.scaleopt has to be set to zero before the next solve.

In most respects GAMS scaling is hidden from the user. The solution values reported back from a solution algorithm are always reported in the notation of the user. The algorithm's internal representation of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if the values of the options limrow and limcol are positive. In addition, the internal representations will appear in the debugging output from the solution algorithm if the option sysout is set to on.

4.37.3.2 Scaling Variables

The scale factor of a variable is defined using the variable attribute .scale in the following way:

```gams
myvar.scale(i,j) = c;
```

The scale factor c is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Note that there is one scale value for each individual component of a multidimensional variable.

Assume that c is the scale factor of a variable V_u. Assume further, that the variable seen by the algorithm is V_a. Then we have: V_a = V_u/c. This means that each variable as seen by the user is divided by the scale factor.

For example, consider the following code snippet:

```gams
Positive Variables x1, x2;
Equation eq;
eq.. 200*x1 + 0.5*x2 =l= 5;
x1.up = 0.01;
x2.up = 10;
x1.scale = 0.01;
x2.scale = 10;
```
By setting \texttt{x1.scale} to 0.01 and \texttt{x2.scale} to 10, the model seen by the solver is:

\begin{verbatim}
Positive Variables xPrime1, xPrime2;
Equation eq;
eq.. 2*xPrime1 + 5*xPrime2 =l= 5;
xPrime1.up = 1;
xPrime2.up = 1;
\end{verbatim}

Note that the solver does not see the variables \texttt{x1} or \texttt{x2}, but rather the scaled (and better-behaved) variables \texttt{xPrime1} and \texttt{xPrime2}. Note further, that upper and lower bounds on variables are automatically scaled in the same way as the variable itself.

**Attention**

- **Discrete variables** cannot be scaled.
- **Expert Note.** Internally, GAMS stores with each variable and equation one additional attribute or field (besides fields for level, marginal and lower and upper bound). Depending on the type of variable and sometimes even model type or solver, this field has different names in the GAMS language. For continuous variables, the field is called \texttt{scale}, while for discrete variable it is called \texttt{prior}. For stochastic 2-stage linear program models solved with DECIS, this field is called \texttt{stage}. The field \texttt{.stage} can lead to confusing results. Consider the following example:

\begin{verbatim}
Variable x;
x.scale = 0.1;
display x.stage;
\end{verbatim}

The output is:

\begin{verbatim}
---- 3 VARIABLE x.scale = 0.100
\end{verbatim}

The field \texttt{.scale} has to be in a certain range ( >1e-20 and no special value), but this is only checked at model generation time. The field \texttt{.prior} can be any number and even \texttt{+inf} (but no other special values). For further information on \texttt{.prior}, see section Setting Priorities for Branching. For an introduction to variable and equation fields, see sections Variable Attributes and Equation Attributes respectively.

### 4.37.3.3 Scaling Equations

The scale factor of an equation is defined using the equation attribute \texttt{.scale} in the following way:

\begin{verbatim}
mzeqn.scale(i,j) = d;
\end{verbatim}

The scale factor \(d\) is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Assume that \(d\) is the scale factor of an equation \(G_u\). Assume further, that the equation seen by the algorithm is \(G_a\). Then we have: \(G_a = G_u/d\). This means that each equation as seen by the user is divided by the scale factor.

For example, consider the following equations:

\begin{verbatim}
Positive Variables y1, y2;
Equations eq1, eq2;
eq1.. 200*y1 + 100*y2 =l= 500;
eq2.. 3*y1 - 4*y2 =g= 6;
\end{verbatim}
By setting `eq1.scale` to 100, the model seen by the solver is:

```
Positive Variables y1, y2;
Equations eqPrime1, eq2;
  eqPrime1.. 2*y1 + 1*y2 =l= 5;
  eq2.. 3*y1 - 4*y2 =g= 6;
```

Note

The user may have to perform a combination of equation and variable scaling to obtain a well-scaled model.

Consider the following example:

```
Positive variables x1, x2;
Equations eq1, eq2;
  eq1.. 100*x1 + 5*x2 =g= 20;
  eq2.. 50*x1 - 10*x2 =l= 5;
x1.up = 0.2;
x2.up = 1.5;
```

Setting the following scale values:

```
x1.scale = 0.1;
eq1.scale = 5;
eq2.scale = 5;
```

will result in the solver seeing the following well-scaled model:

```
Positive Variables xPrime1, x2;
Equations eqPrime1, eqPrime2;
  eqPrime1.. 2*xPrime1 + x2 =g= 4;
  eqPrime2.. xPrime1 - 2*x2 =l= 1;
xPrime1.up = 2;
x2.up = 1.5;
```

### 4.37.3.4 Scaling Derivatives

In nonlinear models the derivatives also need to be well-scaled. Assume that the derivatives in the model of the user are denoted by $d(G_u)/d(V_u)$. Assume further, that the derivatives in the scaled model seen by the algorithm are denoted by $d(G_a)/d(V_a)$. Then we have: $d(G_a)/d(V_a) = d(G_u)/d(V_u) \cdot c/d$, where $c$ is the scale factor for the variable and $d$ is the scale factor for the equation.

The user may affect the scaling of derivatives by scaling both the equation and variable involved.
4.37.3.5 Scaling Data

Scaling input data is independent of the model attribute .scaleopt and may contribute considerably towards achieving a well-scaled model. We recommend users to try to define the units of the input data such that the largest values expected for decision variables and their marginals is under a million, if possible.

For example, in US agriculture about 325 million acres are cropped and the corn crop is 9-10 billion bushels per year. When defining production data, we could enter land in 1000’s of acres and all other resources in 1000’s of units. We could also define the corn crop in millions of bushels. The data will be simultaneously scaled, hence if resource endowments are quoted in 1000’s, corn yields are divided by 1000. This scaling results in a corn production variable in the units of millions. Consumption statistics would need to be scaled accordingly. Money units could also be in millions or billions of dollars. Such data scaling generally greatly reduces the disparity of coefficients in the model.

4.37.4 Conic Programming in GAMS

Conic programming models minimize a linear function over the intersection of an affine set and the product of nonlinear cones. The problem class involving second order (quadratic) cones is known as Second Order Cone Programs (SOCP). These are nonlinear convex problems that include linear and (convex) quadratic programs as special cases.

Conic programs allow the formulation of a wide variety of application models, including problems in engineering and financial management. Examples are portfolio optimization, Truss topology design in structural engineering, Finite Impulse Response (FIR) filter design and signal processing, antenna array weight design, grasping force optimization, quadratic programming, robust linear programming and norm minimization problems.

For more information, see References and Links at the end of this section.

4.37.4.1 Introduction to Conic Programming

Conic programs can be thought of as generalized linear programs with the additional nonlinear constraint \( x \in C \), where \( C \) is required to be a convex cone. The resulting class of problems is known as conic optimization and has the following form:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad A x \leq r^c, \\
& \quad x \in [l^x, u^x] \\
& \quad x \in C
\end{align*}
\]

where \( A \in \mathbb{R}^{m \times n} \) is the constraint matrix, \( x \in \mathbb{R}^n \) the decision variable and \( c \in \mathbb{R}^n \) the objective function cost coefficients. The vector \( r^c \in \mathbb{R}^m \) represents the right-hand side and the vectors \( l^x, u^x \in \mathbb{R}^n \) are lower and upper bounds on the decision variable \( x \).

Now partition the set of decision variables \( x \) into sets \( S^t, t = 1, \ldots, k \), such that each decision variables \( x \) is a member of at most one set \( S^t \). For example, we could have

\[
S^1 = (x_1, x_4, x_7) \quad \text{and} \quad S^2 = (x_6, x_5, x_3, x_2).
\]

Let \( x_{S^t} \) denote the variables \( x \) belonging to set \( S^t \). Then define

\[
C := \{ x \in \mathbb{R}^n : x_{S^t} \in C_t, t = 1, \ldots, k \},
\]

where \( C_t \) must have one of the following forms:
4.37 Special Features for Mathematical Programs

- **Quadratic cone** (also referred to as Lorentz or ice cream cone):

\[
C_t = \left\{ x \in \mathbb{R}^n : x_1 \geq \sqrt{\sum_{j=2}^{n} x_j^2} \right\}.
\]

- **Rotated quadratic cone** (also referred to as hyperbolic constraints):

\[
C_t = \left\{ x \in \mathbb{R}^n : 2x_1 x_2 \geq \sum_{j=3}^{n} x_j^2, x_1, x_2 \geq 0 \right\}.
\]

These two types of cones allow the formulation of quadratic, quadratically constrained and many other classes of nonlinear convex optimization problems.

4.37.4.2 Implementation of Conic Constraints in GAMS

The recommended way to write conic constraints is by using a quadratic formulation. Many solvers have the capability to identify the conic constraints in a QCP model even if it is not in perfect form but can be easily reformulated to fit in the described form. However, some solvers (namely MOSEK) expect the conic constraints to be precisely in the form given above. In earlier versions this form was enforced by a special type of equation, the $\equiv=$ equation type. Moreover, such solvers have other requirements (e.g. disjunctive cones) that can be easily fulfilled by simple reformulation steps. Much progress is expected on the solver side in the coming years, so we don't go into much detail here.

Observe that we could formulate conic problems as regular NLPs using the following constraints:

- **Quadratic cone:**

\[
x('1') \equiv \sqrt{\text{sum}(i[\text{not sameas}(i, '1')], \text{sqr}[x(i)])};
\]

- **Rotated quadratic cone:**

\[
2x('1')x('2') \equiv \text{sum}(i[\text{not sameas}(i, '1') \text{ and not sameas}(i, '2')], \text{sqr}[x(i)]);
\]

Here \(x('1')\) and \(x('2')\) are positive variables.

The following example illustrates the different formulations for conic programming problems. Note that a conic optimizer usually outperforms a general NLP method for the reformulated (NLP) cone problems.

4.37.4.3 Example

Consider the following example, which illustrates the use of rotated conic constraints. We will give reformulations of the original problem in regular NLP form and in conic form (with conic constraints).

The original problem is:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} d_i / x_i \quad (4.2) \\
\text{subject to} & \quad a x \leq b \quad (4.3) \\
& \quad x_i \in [l_i, u_i], \quad i = 1, \ldots, n \quad (4.4)
\end{align*}
\]

where \(x \in \mathbb{R}^n\) is the decision variable, \(d, a, l, u \in \mathbb{R}^n\) are parameters with \(l_i > 0\) and \(d_i \geq 0\) and \(b \in \mathbb{R}\) is a scalar parameter. The original model may be written in GAMS using the equations:
We can write an equivalent QCP formulation by using the substitution \( t_i = 1/x_i \) in the objective function and adding a constraint. As we are dealing with a minimization problem, \( d_i \geq 0 \) and \( x_i \geq l_i > 0 \), we can relax the equality \( t_i x_i = 1 \) into an inequality \( t_i x_i \geq 1 \), which results in an equivalent problem with a convex feasible set:

\[
\text{minimize} \sum_{i=1}^{n} d_i t_i \quad (4.5)
\]

subject to

\[
a x \leq b \quad (4.6)
\]

\[
t_i x_i \geq 1, \quad i = 1, \ldots, n \quad (4.7)
\]

\[
x \in [l, u], \quad (4.8)
\]

\[
t \geq 0, \quad (4.9)
\]

where \( t \in \mathbb{R}^n \) is a new decision variable. The GAMS formulation of this QCP is:

\[
\text{defobjc..} \quad \text{sum}(n, d(n)*t(n)) =e= \text{obj} ;
\]

\[
\text{e1..} \quad \text{sum}(n, a(n)*x(n)) =l= b ;
\]

\[
\text{coneqc}(n) \quad t(n)*x(n) =g= 1 ;
\]

Model cqcp /defobjc, e1, coneqc/ ;

t.lo(n) = 0 ;
\]

\[
x.lo(n) = l(n) ;
\]

\[
x.up(n) = u(n) ;
\]

Note that the constraints \( t_i x_i \geq 1 \) are almost in rotated conic form. If we introduce a variable \( z \in \mathbb{R}^n \) with \( z_i = \sqrt{2} \), then we can reformulate the problem using conic constraints as:

\[
\text{minimize} \sum_{i=1}^{n} d_i t_i \quad (4.11)
\]

subject to

\[
a x \leq b \quad (4.12)
\]

\[
z_i = \sqrt{2}, \quad i = 1, \ldots, n \quad (4.13)
\]

\[
2t_i x_i \geq z_i^2, \quad i = 1, \ldots, n \quad (4.14)
\]

\[
x \in [l, u], \quad (4.15)
\]

\[
t \geq 0, \quad (4.16)
\]

The GAMS formulation using conic equations is:

\[
\text{defobjc..} \quad \text{sum}(n, d(n)*t(n)) =e= \text{obj} ;
\]

\[
\text{e1..} \quad \text{sum}(n, a(n)*x(n)) =l= b ;
\]

\[
\text{e2(n)} \quad z(n) =e= \text{sqr}(2) ;
\]

\[
\text{coneperfect(n)} \quad 2*x(n)*t(n) =g= \text{sqr}(z(n)) ;
\]

Model cperfect /defobjc, e1, e2, coneperfect/ ;

t.lo(n) = 0 ;
\]

\[
x.lo(n) = l(n) ;
\]

\[
x.up(n) = u(n) ;
\]
4.37 Special Features for Mathematical Programs

The complete model is listed below:

Set n / n1*n10 /;
Parameter d(n), a(n), l(n), u(n);
Scalar b;

d(n) = uniform(1,2);
a(n) = uniform(10,50);
l(n) = uniform(0.1,10);
u(n) = l(n) + uniform(0,12-l(n));

Variables x(n);
x.l(n) = uniform(l(n), u(n));
b = sum(n, x.l(n)*a(n));

Variables t(n), z(n), obj;
Equations defobjc, defobj, e1, e2(n), coneqcp(n), coneperfect(n), conenlp(n);

defobjc.. sum(n, d(n)*t(n)) =e= obj;
defobj.. sum(n, d(n)/x(n)) =e= obj;
e1.. sum(n, a(n)*x(n)) =l= b;
coneqcpcp(n).. t(n)*x(n) =g= 1;
e2(n).. z(n) =e= sqrt(2);
coneperfect(n).. 2*x(n)*t(n) =g= sqr(z(n));

Model cqcp /defobjc, e1, coneqcpcp/;
Model cperfect /defobjc, e1, e2, coneperfect/;
Model orig /defobj, e1/;

t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);

Option qcp=cplex;
Solve cqcp min obj using qcp;
Option qcp=mosek;
Solve cperfect min obj using qcp;
Solve orig min obj using nlp;

4.37.4.4 Sample Conic Models in GAMS

Conic models in the GAMS model library include:

- [EMFL]: A multiple facility location problem,
- [FDESIGN]: Linear Phase Lowpass Filter Design,
- [IMMUN]: Financial Optimization: Risk Management,
- [PMEANVAR]: Mean-Variance Models with variable upper and lower Bounds,
- [QP7]: A portfolio investment model using rotated quadratic cones (quadratic program using a Markowitz model),
- [ROBUSTLP]: Robust linear programming as an SOCP,
- [SPRINGCHAIN]: Equilibrium of System with Piecewise Linear Spring,
- [TRUSSM]: Truss Toplogy Design with Multiple Loads
4.37.4.5 References and Links

- Seventh Dimacs Implementation Challenge on Semidefinite and Related Optimization Problems.

4.37.5 Indicator Constraints

An indicator constraint is a way of expressing relationships between variables by specifying a binary variable to control whether or not a constraint takes effect. For example, indicator constraints are useful in problems where there are fixed charges to express only if a given variable comes into play.

So-called Big M formulations require to estimate an upper bound of an expression in a model. In most cases the model data can be used to determine relatively small number for such coefficients. In some cases it is not possible to find small Big M values and a resulting solution may exhibit trickle flow and have other unwanted side-effects. The main purpose of indicator constraints is to overcome these limitations of Big M formulations. Generally, the use of indicator constraints is not warranted when the unwanted side-effects of Big M formulations are not present.

Consider the following example:

```
constr01.. x1 + x2 + x3 =l= 1e+9*y; // may cause problems
```

Here we use a Big M formulation, that relies on the x values to sum to less than one billion. Note that this formulation may cause numeric instability or undesirable solutions in some situations. Alternatively, we could use an indicator constraint:

```
constr01$(y=0).. x1 + x2 + x3 =l= 0; // alternative
```

Note that y is a binary variable; the logical condition makes sure that the constraint is only active if y = 0. Unfortunately, the $() expressions do not allow endogenous variables and we need a different way to specify such implications. Indicators are supported by CPLEX, GUROBI, SCIP, and XPRESS.

We should mention a formulation of indicators without the explicit indicator constraints. We can use a SOS1 constraint to express that a constraint holds or not based on a binary variable by having a SOS1 set of a slack variable and the binary variable SOS1(slack,1-y):

```
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack; // alternative
```

Recall that only one of the variables in a SOS1 constraint can be non-zero, so if 1-y is non-zero (i.e. y=0) then slack must be zero, i.e. the constraint holds. If 1-y=0 is non-zero, i.e. y=1 then slack can be used to make the constraint feasible for any setting of x1, x2, and x3. Here two unrelated variable, slack and y are part of an SOS1 constraint and some tricks are required to formulate this properly in GAMS:

```
Set oo / slack, yExpr /;
SOS1 variable indic(oo);
Equation iOn, iOff;
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack; // alternative
```

We should mention a formulation of indicators without the explicit indicator constraints. We can use a SOS1 constraint to express that a constraint holds or not based on a binary variable by having a SOS1 set of a slack variable and the binary variable SOS1(slack,1-y):

```
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack; // alternative
```
4.37.5.1 Indicator Constraints with GAMS

In the remainder of this section we will describe how to specify indicator constraints when using GAMS. Please consult the corresponding solver manual for information on whether indicator constraints are supported and possible differences in their specification.

The example from above will be implemented with a constraint in combination with additional information in a solver option file, e.g. CPLEX: cplex.opt:

```plaintext
constr01.. x1 + x2 + x3 =l= 0;
```

and the following entry in the option file:

```plaintext
indic constr01$y 0
```

This has the following effect: equation constr01 will become an indicator constraint and becomes active in a solution where the binary variable takes the value 0. If the value of $y$ in a solution is 1, the constraint is not active.

Note that this way of entering an indicator constraint is dangerous since the option files changes the model (usually an option file has some effect on the performance of the algorithm). Therefore, the solver will abort if there is a problem processing the indicator options in the solver option file.

Attention

If the model is given to a solver without the option file containing the indicator mapping (or to a solver that does not understand the `indic` keyword, a very different model will be solved. The current implementation of indicator constraints requires a significant amount of caution from the user.

There are two ways of entering the equation/binary variable mapping in a solver option file: with an indexed format and using labels.

The indexed format is a convenient shorthand notation which borrows its syntax from the GAMS syntax. It requires that the indices for the binary variable are already present in the index set of the equation.

Consider the following example of an invalid GAMS syntax with an endogenous variable in the dollar condition:

```plaintext
equ1(i,j,k)$(ord(i) < ord(j) and bin1(i,k)=1).. lhs =l= rhs;
```

This may be specified with the following equation in the GAMS file:

```plaintext
equ1(i,j,k)$(ord(i) < ord(j)).. lhs =l= rhs;
```

plus a solver option file with the following entry:

```plaintext
indic equ1(i,j,k)$bin1(i,k) 1
```
The *label format* is used in cases where the binary variable indices are not present in the equation indices or the binary variable is adjusted with lags or leads. In these cases the mapping of all individual equations and variables of the indicator constraints need to be specified. An example follows.

Set  
    i /i1*i3/, j /j1*j2/;
Binary variable bin1(j);
Equation  
    equ1(i,j);

\[
equ1(i,j)\$(bin1(j++1)=0).. lhs =e= 0;
\]

Note that the example above is not valid GAMS code. Instead, we will combine a valid GAMS equation and a solver option file using the label format as follows:

\[
equ1(i,j).. lhs =e= 0;
\]

and the solver option file

\[
indic equ1('i1','j1')$bin1('j2') 0
indic equ1('i1','j2')$bin1('j1') 0
indic equ1('i2','j1')$bin1('j2') 0
indic equ1('i2','j2')$bin1('j1') 0
indic equ1('i3','j1')$bin1('j2') 0
indic equ1('i3','j2')$bin1('j1') 0
\]

Note that the lines in such option files need not be entered manually. They may be easily generated using the GAMS The Put Writing Facility. For example, the lines above may be generated as follows:

```
file fcpx / cplex.opt /;
fcpx.pc=8;
loop((i,j), put fcpx 'indic' equ1(i,j) '$' bin1(j++1) '0'/);
putclose fcpx;
```

There are situations where the indicator binary variable exist in the indicator constraint only and hence will not be generated by GAMS to be passed on to the solver. In such cases the solver will issue the following error message:

**Error: Column is not of type Variable**

There is an easy way to fix this problem: adding the binary indicator variable artificially to the model. For example, it may be added with the coefficient \(\varepsilon\) to the objective:

```
defobj.. z =e= ... + \varepsilon*sum(j, bin1(j));
```
4.37.5.2 An Example for Indicator Constraints with GAMS

In this subsection we will comment on parts of the model [TRNSINDIC]. This model uses big M formulations and indicator constraints to solve the same problem. In addition, a formulation that makes it easy to switch between these two is presented. It is a fixed-charge network example based on the well-known model [TRNSPORT].

Recall that $i$ is the set of canning plants and $j$ is the set of markets where cases of some product are to be shipped. First, the basic model is reformulated to a MIP by introducing the binary variable $\text{use}(i,j)$ and two new equations:

Binary Variable $\text{use}(i,j)$ is 1 if arc is used in solution;

Equations

- $\text{minship}(i,j)$ ensure minimum shipping
- $\text{maxship}(i,j)$ ensure zero shipping if use variable is 0;

\[
\begin{align*}
\text{minship}(i,j) & : \quad x(i,j) \geq \text{minshipping} \times \text{use}(i,j); \\
\text{maxship}(i,j) & : \quad x(i,j) \leq \text{bigM} \times \text{use}(i,j);
\end{align*}
\]

Note that $\text{minshipping}$ is a scalar denoting the minimum amount of cases that may be shipped and $\text{bigM}$ is a sufficiently large number, as usual.

Next, the same problem is solved with indicator constraints: the two new equations are reformulated and a CPLEX option file with information on indicator constraints is added:

Equations

- $\text{iminship}(i,j)$ ensure minimum shipping using indicator constraints
- $\text{imaxship}(i,j)$ ensure zero shipping if use variable is 0 using indicator constraints;

\[
\begin{align*}
\text{iminship}(i,j) & : \quad x(i,j) \geq \text{minshipping}; \\
\text{imaxship}(i,j) & : \quad x(i,j) = 0;
\end{align*}
\]

Model $\text{indicatorModel} /\text{cost}, \text{supply}, \text{demand}, \text{iminship}, \text{imaxship}/$;

file fcpx Cplex Option file / cplex.opt /;
putclose fcpx 'indic iminship(i,j)$use(i,j) 1' / 'indic imaxship(i,j)$use(i,j) 0';
indicatorModel.optfile = 1;
Solve indicatorModel using mip minimizing $z$;

Note that the option file contains an entry for each of the two equations. Note further, that the binary variable $\text{use}$ moved from the equations to the option file. However, it also features in the objective equation of the model, therefore this is not problematic. Observe that the indexed format for the equation/binary variable mapping was used in the option file. Alternatively, the label format may be used:

\[
\begin{align*}
\text{loop}((i,j), \\
\quad \text{put fcpx 'indic ' iminship.tn(i,j)$use.tn(i,j) yes} \\
\qquad / 'indic ' imaxship.tn(i,j)$use.tn(i,j) no / );
\end{align*}
\]

In a final step the model is reformulated again such that the same problem may be solved with and without indicator constraints. This can be especially useful for debugging a model with indicator constraints. For details, see [TRNSINDIC].

Another example of a model with indicator constraints is the model [BILINEAR] where various formulations to represent bilinear product terms are demonstrated.
4.38 GAMS Output

4.38.1 Introduction

The output from GAMS contains many components in support for checking and comprehending a model. This chapter discusses the components of the GAMS output file generated from a GAMS run as well as ways to control the amount of diagnostic output produced. A GAMS run also generates a log which serves as a useful first step in analysing a GAMS run. The details regarding the log can be found in the GAMS log chapter.

The output file generated from a GAMS run is called *listing file*. The listing file has the file extension `.lst` and can be read using any text editor. By default the listing file has the same file name as the input file, but this can be changed using the command line parameter `Output`. See chapter The GAMS Call and Command Line Parameters for more information. The main components in the listing file are:

1. **Compilation.** The compilation output contains an echo print of the input file, possibly error messages, along with lists of GAMS objects and cross reference maps.
2. **Execution.** The execution output contains the results of display statements and possibly execution error messages.
3. **Model Generation.** The output generated during model generation contains listings of equations and variable listings as well as model statistics and possibly generation execution error messages.
4. **Solution.** The output generated when an external solver program processes the model is the solution report including the solve summary, the solver report, the solution listing and the report summary.
5. **Post-Solution.** The final components added to the listing file are the final execution summary and the file summary.

A small nonlinear program [ALAN] will be used to illustrate every component of the listing file. The possibilities for extension to large models with voluminous output (which is when the diagnostics are really useful) should be apparent. After covering all the components of the listing file we will discuss error reporting and conclude the chapter with a compiled list of how to customize the output file.

This chapter does not cover the output that can be generated by the user through display statements and put statements. This output is covered in the chapters The Display Statement and The Put Writing Facility respectively.

4.38.2 An Illustrative Model

[ALAN] is a portfolio selection model whose objective is to choose a portfolio of investments whose expected return meets a target while minimizing the variance. We will discuss a simplified version of this model. The input file is listed for reference.

$Title A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
$onsymlist onsymxref onuellist onuelxref
$Ontext
This is a mini mean-variance portfolio selection problem described in 'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
Set i  securities /hardware, software, show-biz, t-bills/;
alias (i,j);

Scalar target  target mean annual return on portfolio % /10/,
   lowyield  yield of lowest yielding security,
   highrisk variance of highest security risk ;

Parameters mean(i) mean annual returns on individual securities (%)
    / hardware 8
      software 9
      show-biz 12
      t-bills  7 /

Table v(i,j) variance-covariance array (%-squared annual return)
    hardware software show-biz t-bills
    hardware 4 3 -1 0
    software 3 6 1 0
    show-biz  1 1 10 0
    t-bills  0 0 0 0 ;

lowyield = smin(i, mean(i)) ;
highrisk = smax(i, v(i,i)) ;
display lowyield, highrisk ;

Variables x(i) fraction of portfolio invested in asset i
   variance variance of portfolio
Positive Variable x;

Equations fsum  fractions must add to 1.0
    dmean definition of mean return on portfolio
    dvar  definition of variance;

fsum..  sum(i, x(i))       =e=  1.0;
dmean.. sum(i, mean(i)*x(i)) =e=  target;
dvar.. sum(i, x(i)*sum(j,v(i,j)*x(j))) =e=  variance;

Model portfolio / fsum, dmean, dvar / ;
Solve portfolio using nlp minimizing variance;
display x.l, variance.l;

4.38.3 Compilation Output

The compilation output is the output produced during the initial check of the program, often referred to as compilation. It contains the following parts: the echo print of the input file, the symbol reference map, the symbol listing map, the unique element listing map, and the include file summary.

By default only the echo print and the include file summary is shown, the other parts are suppressed and may be turned on with dollar control options. See also section Customizing the Output File on how to control the appearance and amount of detail in the output file produced by the GAMS compiler.
4.38.3.1 The Echo Print of the Input File

The *echo print* of the program is always the first part of the output file. It is a listing of the input with added line numbers. It is possible to control the listing output using `$offlisting` to suppress the echo print and `$onlisting` to turn the echo print on again. The first few lines of the echo print of the Illustrative Model follow:

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)

Compilation

2

This is a mini mean-variance portfolio selection problem described in 'GAMS/MINOS: Three examples' by Alan S. Manne, Department of Operations Research, Stanford University, May 1986.

10

* This model has been modified for use in the documentation

The default header of the GAMS listing file is General Algebraic Modeling System. This may be replaced using the $title at the start of a line. The text on that line will be the new header, as demonstrated above. After the header the compilation output is announced with the title Compilation.

The first three line numbers refer to empty lines in the input and line 11 is a comment line. If the lines on the input file are counted, it can be seen that this comment line appears after ten lines of dollar directives, comments and empty lines.

The dollar control options that follow in the model are used to display more information in the output file than the default and will be discussed in the subsections below.

Note

By default, dollar control option lines are not listed in the echo print. The $onDollar and $offDollar controls the echoing of Dollar Control Options lines in the listing file. Dollar control option lines are also listed if they contain errors.

The block comment enclosed by $ontext and $offtext is listed without line numbers, while single line comments starting with asterisks (*) are listed with the respective line numbers. Observe that line numbers always refer to the physical line number in the input file. The remainder of the echo print follows:

12
13 Set i securities /hardware, software, show-biz, t-bills/;
14 alias (i,j);
15
16 Scalar target target mean annual return on portfolio % /10/,
17 lowyield yield of lowest yielding security,    
18 highrisk variance of highest security risk ;
19
20 Parameters mean(i) mean annual returns on individual securities (%)
21 / hardware 8
22 software 9
23 show-biz 12
24 t-bills 7 / ;
25
26 Table v(i,j) variance-covariance array (%-squared annual return)
### Symbol Listing

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TYPE</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmean</td>
<td>EQU</td>
<td>declared</td>
</tr>
<tr>
<td>dvar</td>
<td>EQU</td>
<td>declared</td>
</tr>
<tr>
<td>fsum</td>
<td>EQU</td>
<td>declared</td>
</tr>
<tr>
<td>highrisk</td>
<td>PARAM</td>
<td>declared</td>
</tr>
<tr>
<td>i</td>
<td>SET</td>
<td>declared</td>
</tr>
<tr>
<td>j</td>
<td>SET</td>
<td>declared</td>
</tr>
</tbody>
</table>

Lines in the echo print can be set to double space using `$double` and then reset to single space with `$single`. If errors were detected, the explanatory messages would be found at the end of the echo print. All discussions of error messages have been grouped together in the section **Error Reporting** below.

#### 4.38.3.2 The Symbol Reference Map

The other parts of the compilation outputs are maps. They are extremely useful for users looking at a model written by someone else or trying to make changes in their own model after spending some time away from it. By default all maps are suppressed and can be turned on with dollar control options that are specified below.

The first map is the **symbol cross reference**. It can be turned on with the dollar control option `$onSymXRef` at the beginning of the program. The symbol cross reference map lists all identifiers (symbols) from the model in alphabetical order, identifies them as to data type, shows the line numbers where the symbols appear and classifies each appearance. The map that was generated as part of the output of our *Illustrative Model* is shown below:
In the first two columns the name and type of each identifier are given. For example, the last symbol listed is x which is defined to be of type VAR, a shorthand symbol for variable. The complete list of data types and their shorthand symbols is given below:

<table>
<thead>
<tr>
<th>Shorthand Symbol</th>
<th>GAMS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACRNM</td>
<td>acronym</td>
</tr>
<tr>
<td>EQU</td>
<td>equation</td>
</tr>
<tr>
<td>FILE</td>
<td>put file</td>
</tr>
<tr>
<td>MODEL</td>
<td>model</td>
</tr>
<tr>
<td>PARAM</td>
<td>parameter</td>
</tr>
<tr>
<td>SET</td>
<td>set</td>
</tr>
<tr>
<td>VAR</td>
<td>variable</td>
</tr>
</tbody>
</table>

For further details on data types in GAMS, see section Data Types and Definitions.

The symbol highrisk was declared as a scalar, but is listed as a parameter in the map above. Recall that parameters, scalars, and tables are three input formats for the data type parameter. For more information, see the overview Parameters, Scalars and Tables.

After the name and type of the identifier, a list of references to the symbol is given. References are grouped by reference type and identified by the line number in the output file. The actual reference can be found by referring to the echo print of the program, which has line numbers on it. In the case of the symbol x in the example above, the list of references as shown in the symbol reference map are as follows:

declared 37
impl-asn 50
ref 39 45 46 2*47 51

This means that x is declared on line 37, implicitly assigned through a solve statement on line 50 and referenced on lines 39, 45, 46, 47, and 51. The entry 2*47 means that there are two references to x on line 47 of the input file.

The complete list of reference types and their shorthand symbols in GAMS Output is given below:

<table>
<thead>
<tr>
<th>Shorthand Symbol</th>
<th>Reference Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declared</td>
<td>Declaration</td>
<td>The identifier is declared as to type. This must be the first appearance of the identifier.</td>
</tr>
<tr>
<td>defined</td>
<td>Definition</td>
<td>An initialization (for a table or a data list between slashes) or symbolic definition (for an equation) starts for the identifier.</td>
</tr>
<tr>
<td>assigned</td>
<td>Assignment</td>
<td>Values are replaced because the identifier appears on the left-hand side of an assignment statement.</td>
</tr>
<tr>
<td>impl-asn</td>
<td>Implicit Assignment</td>
<td>An equation or variable will be updated as a result of being referred to implicitly in a solve statement.</td>
</tr>
<tr>
<td>Shorthand Symbol</td>
<td>Reference Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>control</td>
<td>Control</td>
<td>A set is used as (part of) the driving index in an assignment, equation, loop or indexed operation.</td>
</tr>
<tr>
<td>ref</td>
<td>Reference</td>
<td>The symbol has been referenced on the right-hand side of an assignment or in a display, equation, model, solve statement or put statement.</td>
</tr>
<tr>
<td>index</td>
<td>Index</td>
<td>Like control, but used only for set labels. Appears only in the cross reference map of unique elements. See section The Unique Element Listing Map for details.</td>
</tr>
</tbody>
</table>

Note that the symbol reference map may be useful in model development, documentation preparation, and to make sure that all declared items are actually used in the model (by checking that they all have reference symbols in addition to declared and defined).

### 4.38.3.3 The Symbol Listing Map

The second optional map is the symbol listing. In the symbol listing map all identifiers are grouped alphabetically by data type and listed with their explanatory texts. This is another very useful aid for trying to understand a large model prepared by someone else. Note that expressive explanatory text is particularly helpful here. The symbol listing map can be turned on by entering a line containing the dollar control option $onSymList at the beginning of the program. The symbol listing map generated from our Illustrative Model follows:

**SETS**

- i securities
- j Aliased with i

**PARAMETERS**

- highrisk variance of highest security risk
- lowyield yield of lowest yielding security
- mean mean annual returns on individual securities (%)
- target target mean annual return on portfolio %
- v variance-covariance array (%-squared annual return)

**VARIABLES**

- variance variance of portfolio
- x fraction of portfolio invested in asset i

**EQUATIONS**

- dmean definition of mean return on portfolio
- dvar definition of variance
- fsum fractions must add to 1.0

**MODELS**

portfolio
4.38.3.4 The Unique Element Listing Map

The last optional map is the Unique Element Listing (UEL). The unique element listing map can be turned on by entering a line containing the dollar control option $onUELList at the beginning of the program. All unique elements are first grouped in entry order and then in sorted order with their explanatory texts:

Unique Element Listing

Unique Elements in Entry Order
1 hardware software show-biz t-bills

Unique Elements in Sorted Order
1 hardware show-biz software t-bills

In addition, a map with references for every label is given if the dollar control option $onUELEXRef appears at the beginning of the program.

ELEMENT REFERENCES

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardware</td>
<td>declared</td>
</tr>
<tr>
<td>show-biz</td>
<td>declared</td>
</tr>
<tr>
<td>software</td>
<td>declared</td>
</tr>
<tr>
<td>t-bills</td>
<td>declared</td>
</tr>
</tbody>
</table>

The unique element list is important in the context of ordered sets. For details see section Ordered and Unordered Sets.

4.38.3.5 The Include File Summary

Often the GAMS program includes a text file specified by the dollar control option $include. Consider the following example:

$include 'file1.inc'
Set i / i1*i10 /
Scalar a; a = 7;
$include 'file2.inc'

The file file1.inc contains the following two lines:

Set j / j1*j5 /
Parameter k(j);

The file file2.inc contains the following line:

a = a+3;
The echo print follows:

```
\texttt{\include C:\models\file1.inc}
2 \texttt{Set j / j1\*j5 /;}
3 \texttt{Parameter k(j);}   
4 \texttt{Set i / i1\*i10 /;}
5 \texttt{Scalar a; a = 7;}
\texttt{\include C:\models\file2.inc}
7 \texttt{a = a+3;}
```

Note that the contents of the two include files are expanded in the echo print and the include file names and their path is echoed in the lines with the dollar control option $\texttt{include}$. If we insert the dollar control option $\texttt{offInclude}$ in the first line of the code, the echo print will not include the lines with the names of the include files any more, but the contents will still be echoed:

```
3 \texttt{Set j / j1\*j5 /;}
4 \texttt{Parameter k(j);}   
5 \texttt{Set i / i1\*i10 /;}
6 \texttt{Scalar a; a = 7;}
8 \texttt{a = a+3;}
```

If the program contains include files, there will be an Include File Summary at the end of the compilation output. The include file summary of the simple example above (without the line $\texttt{offInclude}$) follows. If $\texttt{offInclude}$ is active, it will be suppressed.

<table>
<thead>
<tr>
<th>SEQ</th>
<th>GLOBAL</th>
<th>TYPE</th>
<th>PARENT</th>
<th>LOCAL</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>INPUT</td>
<td>0</td>
<td>0</td>
<td>C:\models\test.gms</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>INCLUDE</td>
<td>1</td>
<td>1</td>
<td>.C:\models\file1.inc</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>INCLUDE</td>
<td>1</td>
<td>4</td>
<td>.C:\models\file2.inc</td>
</tr>
</tbody>
</table>

In the first column the sequence number of the input files is given. Note that the parent file called by the GAMS call is always listed first. The second column gives the global (expanded) line number which contains the respective option include or one of its variants. The third column indicates the type of the respective file. An overview of the file types is given below:

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>The GAMS input file that was called with the GAMS call.</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>A file that was inserted with the dollar command option $\texttt{include}$.</td>
</tr>
<tr>
<td>BATINCLUDE</td>
<td>A file that was inserted with the dollar command option $\texttt{batinclude}$.</td>
</tr>
<tr>
<td>LIBINCLUDE</td>
<td>A file that was inserted with the dollar command option $\texttt{libinclude}$.</td>
</tr>
<tr>
<td>SYSINCLUDE</td>
<td>A file that was inserted with the dollar command option $\texttt{sysinclude}$.</td>
</tr>
</tbody>
</table>

The next column, named PARENT, provides the sequence number of the respective parent file. The column with the header LOCAL gives the local line number in the parent file where the file was included. The last column shows the path and the name of the respective file.

Note that compilation error messages in include files have additional information about the name of the include file and the local line number. For information on compilation errors, see section Compilation Errors below.
4.38.4 Execution Output

The execution output follows the compilation output in the GAMS listing file. It is introduced with the title **Execution**. The execution output is the output generated while GAMS is performing data manipulations and results from display statements. The output from the display statement on line 33 of our **Illustrative Model** is shown below.

```
---  33 PARAMETER lowyield = 7.000 yield of lowest yielding security
      PARAMETER highrisk = 10.000 variance of highest security risk
```

If errors are detected because of illegal data operations, a brief message indicating the cause and the line number of the offending statement will appear. For further information on execution errors, see section **Execution Errors** below.

**Note**

In case there is no display statement before The Solve Statement in the model and there are no execution errors, there will be no execution output in the GAMS listing file.

4.38.5 Model Generation Output

The model generation output, the solution report, and the post-solution Output are produced when a solve statement is executed. The actions that are initiated by a solve statement are presented in section **Actions Triggered by the Solve Statement**. All output generated as a result of a solve is labeled with a subtitle identifying the model, its type and the line number of the solve statement.

The output generated during model generation includes equations listing, variable listing, and model statistics.

4.38.5.1 The Equation Listing

The **equation listing** is the first part of the output generated by a solve statement. It is marked with the subtitle **Equation Listing**. By default, the first three equations in every block are listed. This can be modified with the option limrow. If there are three or fewer single equations in any equation block, then all the single equations are listed.

Note that by studying the equation listing the user may determine whether the model generated by GAMS is the model that the user has intended - an extremely important question. This component of the output shows the specific equations generated within the model when the current values of the sets and parameters are plugged into the general algebraic form of the model.

The equation listing from our **Illustrative Model** is given below. In our case we have three equation blocks, each producing one single equation.
A Quadratic Programming Model for Portfolio Analysis (ALAN, SEQ=124a)
Equation Listing  SOLVE portfolio Using NLP From line 50

---- fsum  =E=  fractions must add to 1.0
fsum..  x(hardware) + x(software) + x(show-biz) + x(t-bills) =E= 1 ; (LHS = 0, INFES = 1 ****)

---- dmean  =E=  definition of mean return on portfolio
dmean..  8*x(hardware) + 9*x(software) + 12*x(show-biz) + 7*x(t-bills) =E= 10 ; (LHS = 0, INFES = 10 ****)

---- dvar  =E=  definition of variance
dvar..  (0)*x(hardware) + (0)*x(software) + (0)*x(show-biz) - variance =E= 0 ; (LHS = 0)

Note
The equation listing is an extremely useful debugging aid. It shows the variables that appear in each constraint. In addition, it shows what the individual coefficients and right-hand side values evaluate to after the data manipulations have been done.

Each equation block is marked with four dashes which are useful for mechanical searching. The name, type of equation and explanatory text is shown, followed by the individual equations.

Note
All the terms that depend on variables are collected on the left-hand side and all the constant terms are combined into one number on the right-hand side, any necessary sign changes being made.

Four places of decimals are shown if necessary, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers to be displayed as zeros.

Note
Nonlinear equations are treated differently. If the coefficient of a variable in the equation listing is enclosed in parentheses, then the corresponding constraint is nonlinear, and the value of the coefficient depends on the activity levels of one or more of the variables. The listing is not algebraic, but shows the partial derivative of each variable evaluated at their current level values.

Note that in the equation listing from our example the equation dvar is nonlinear. A simpler example will help to clarify the point. Consider the following equation and associated level values.

eq1..  2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2; x.l = 2; y.l = 3 ;

This equation will appear in the equation listing as:

eq1.. (221)*x + (216.1667)*y =E= 2 ; (LHS = 225.5 ***)
The coefficient of $x$ is determined by first differentiating the equation above with respect to $x$. This results in $2 \times (2 \times x.l) \times (y.l)^3 + 5$, which evaluates to 221, given that $x.l$ equals 2 and $y.l$ equals 3. Similarly, the coefficient of $y$ is obtained by differentiating the equation above with respect to $y$, which results in $2 \times (x.l)^2 \times 3 \times (y.l)^2 + \frac{1.5}{(y.l)^2}$, giving 216.1667. Observe that the coefficient of $y$ could not have been determined if its level had been left at zero. The attempted division by zero would have produced an error and premature termination. For further information on modeling NLP problems with GAMS, see the tutorial Good NLP Formulations.

The result of evaluating the left-hand side of the equation at the initial point is shown at the end of each individual equation listing. In the example above it is 225.5, and the three asterisks(*** are a warning that the constraint is infeasible at the starting point.

Note

The order in which the equations are listed depends on how the model was defined. If it was defined with a list of equation names, then the listing will be in the order of that list. If it was defined as /all/, then the list will be in the order of declaration of the equations. The order of the entries for the individual constraints is determined by the label entry order.

### 4.38.5.2 The Column Listing

The column listing or variable listing is the next part of the output. It is marked with the title Column Listing and contains a list of the individual coefficients sorted by column rather than by row (like in the equation listing). By default the first three entries for each variable are shown, along with their lower bound .lo, upper bound .up and current level values .l. Note that the default number of entries shown may be modified with the option limcol.

The format for the coefficients is exactly as in the equation listing, with the nonlinear coefficients enclosed in parentheses and the trailing zeroes dropped. The column listing from our Illustrative Model follows.

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)

Column Listing SOLVE portfolio Using NLP From line 50

---- x fraction of portfolio invested in asset i

x(hardware)   (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  1   fsum
  8   dmean
  (0) dvar

x(software)   (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  1   fsum
  9   dmean
  (0) dvar

x(show-biz)   (.LO, .L, .UP, .M = 0, 0, +INF, 0)
  1   fsum
 12   dmean
  (0) dvar

REMAINING ENTRY SKIPPED
4.38 GAMS Output

---- variance  variance of portfolio

variance

(.LO, .L, .UP, .M = -INF, 0, +INF, 0)

-1  dvar

Note
The order in which the variables appear is the order in which they were declared.

4.38.5.3 The Range Statistics

This block shows a statistic about the range of values seen in the model. Namely, it prints the range of absolute non-zero values for the bounds, right hand sides and matrix coefficients. Zero values are marked explicitly. This is the example output when running the model [TRNSPORT]:

Range Statistics  SOLVE transport Using LP From line 71

RANGE STATISTICS (ABSOlUTE NON-ZERO FINITE VALUES)

RHS  [min, max] : [ 2.750E+02, 6.000E+02] - Zero values observed as well
Bound  [min, max] : [ NA, NA] - Zero values observed as well
Matrix  [min, max] : [ 1.260E-01, 1.000E+00]

Note the NA range for the bounds: For this model, the bounds are either 0 or +INF. Both values are excluded in this range calculation. Therefore we see the NA here.

4.38.5.4 The Model Statistics

The final information generated while a model is being prepared for solution is the statistics block. It is marked with the title Model Statistics. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of our Illustrative Model follow:

A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Model Statistics  SOLVE portfolio Using NLP From line 50

MODEL STATISTICS

| BLOCKS OF EQUATIONS | 3 | SINGLE EQUATIONS | 3 |
| BLOCKS OF VARIABLES | 2 | SINGLE VARIABLES | 5 |
| NON ZERO ELEMENTS   | 12 | NON LINEAR N-Z   | 3 |
| CODE LENGTH         | 25 | CONSTANT POOL    | 17 |

GENERATION TIME  = 0.004 SECONDS  4 MB

EXECUTION TIME  = 0.005 SECONDS  4 MB
The **BLOCK** counts indicate the number of GAMS symbols (**equations** and **variables**, respectively) appearing in the problem generated. The **SINGLE** counts indicate the number of individual rows and columns in the problem generated. The **NON ZERO ELEMENTS** entry refers to the number of nonzero coefficients in the problem matrix. To understand the remaining elements, some background will be useful.

For linear models, the model can be completely specified by the problem matrix and vectors for the RHS, etc. More is needed for nonlinear models. GAMS passes the nonlinear expressions defining the model algebra to the solvers in the form of **NL code** - opcode/address pairs in a simple stack-based language. For example, the expression \( \exp(x \cdot y - 1.675) \) results in code similar to this:

```
PUSHV column-index-of-x
MULV column-index-of-y
SUBC index-of-1.675
FUNC1 index-of-exp-function
STOR index-of-row
```

In addition to containing indices referring to the rows and columns of the model, the NL code contains indices of constants used to define the model algebra. These constants are not contained in the NL code directly but rather in a separate pool or list of constants used for the model.

There are three entries that provide additional information about nonlinear models. The **NON LINEAR N-Z** entry refers to the number of nonlinear matrix entries in the model. Nonlinear models may differ in the complexity or size of the nonlinear expressions defining them. For example, \( xy \) is a simpler form of nonlinearity than \( \exp(x \cdot y) \) and will require a shorter list of instructions to define, even though both of these terms result in two nonlinear entries in the matrix (one for \( x \) and one for \( y \)). The **CODE LENGTH** entry indicates the length of the NL code (i.e. number of instructions) required to define the model algebra, while the **CONSTANT POOL** indicates the length of the constant pool that is used by the NL code. In general, the more nonlinear a problem is, the more difficult it tends to be to solve.

The times that follow statistics are also useful. The **GENERATION TIME** is the time used since compilation (syntax check) is finished. This includes the time spent in generating the model. The measurement units are given and represent ordinary clock time on personal computers or central processor usage (CPU) time on other machines. Memory use is given in megabytes.

### 4.38.6 The Solution Report

The **solution report** is the next part of the output. It is marked with the title **Solution Report** and includes the **solve summary**, the **solver report**, the **solution listing**, and the **report summary**.

#### 4.38.6.1 The Solve Summary

The **solve summary** contains details about the solution process and is marked with the title **SOLVE SUMMARY**. The first part of the solve summary is common to all solvers and is discussed in this subsection. The second part of the solve summary is solver specific, it is covered in the subsection **Solver Report** below. The first part of the solve summary for our **Illustrative Model** follows.

---

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report SOLVE portfolio Using NLP From line 50

SOLVE SUMMARY

MODEL portfolio OBJECTIVE variance
```
The entry **MODEL** contains the name of the model being solved, **TYPE** provides the model type of the model, **SOLVER** shows the name of the solver used to solve the model, **OBJECTIVE** gives the name of the objective variable being optimized, **DIRECTION** shows the direction of optimization being performed and the entry **FROM LINE** provides the line number of the solve statement the solve summary refers to.

Note that the four asterisks make it easy to find the solve summary if it is searched for mechanically.

The entries **SOLVER STATUS** and **MODEL STATUS** contain the solver and model status for the problem respectively. Their possible values are given in Model Status table and Solver Status tables below. The entry **OBJECTIVE VALUE** provides the value of the objective function at the termination of the solve. This value is the optimum value for the problem provided the solver and model have the right status.

The entry **RESOURCE USAGE, LIMIT** reports the amount of wall clock time (in seconds) taken by the solver and the upper limit allowed for the solver. Note that the solver will stop as soon as the limit on time usage has been reached. The default limit may be changed with the option reslim. Observe that the option statement `option reslim = x;` - where `x` is the desired limit in wall clock seconds - must be entered in the model before the solve statement.

The entry **ITERATION COUNT, LIMIT** provides the number of iterations used by the solver and the upper limit allowed for the solver. The solver will stop as soon as this limit is reached. The default limit on iterations used is practically infinity. This limit may be changed with the option iterlim. Observe that the option statement `option iterlim = n;` - where `n` is the desired limit of the iterations used - must be entered in the model before the solve statement.

Finally, the entry **EVALUATION ERRORS** reports the number of numerical errors encountered by the solver and the upper limit allowed for the solver. These errors result from numerical problems like division by zero. Note that this is suppressed for LP, RMIP and MIP models since evaluation errors are not applicable for these model types. The default limit on evaluation errors used is zero. This limit may be changed with the option domlim. Observe that the option statement `option domlim = n;` - where `n` is the desired limit of the evaluation errors allowed - must be entered in the model before the solve statement.

<table>
<thead>
<tr>
<th>Value</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OPTIMAL</td>
<td>The solution is optimal, that is, it is feasible (within tolerances) and it has been proven that no other feasible solution with better objective value exists. Note that the latter criterion is not influenced by the optcr and optca options.</td>
</tr>
<tr>
<td>2</td>
<td>LOCALLY OPTIMAL</td>
<td>A local optimum for an NLP has been found. That is, a solution that is feasible (within tolerances) and it has been proven that there exists a neighborhood of this solution in which no other feasible solution with better objective value exists.</td>
</tr>
<tr>
<td>3</td>
<td>UNBOUNDED</td>
<td>The solution is unbounded. This message is reliable if the problem is linear, but occasionally it appears for difficult nonlinear problems that are not truly unbounded, but that lack some strategically placed bounds to limit the variables to sensible values.</td>
</tr>
<tr>
<td>Value</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>INFEASIBLE</td>
<td>The problem has been proven to be infeasible. If this was not intended, something is probably misspecified in the logic or the data.</td>
</tr>
<tr>
<td>5</td>
<td>LOCALLY INFEASIBLE</td>
<td>No feasible point could be found for the NLP problem from the given starting point. It does not necessarily mean that no feasible point exists.</td>
</tr>
<tr>
<td>6</td>
<td>INTERMEDIATE INFEASIBLE</td>
<td>The current solution is not feasible, but the solver stopped, either because of a limit (for example, iteration or resource) or because of some sort of difficulty. The solver status will give more information.</td>
</tr>
<tr>
<td>7</td>
<td>FEASIBLE SOLUTION</td>
<td>A feasible solution to a problem without discrete variables has been found.</td>
</tr>
<tr>
<td>8</td>
<td>INTEGER SOLUTION</td>
<td>A feasible solution to a problem with discrete variables has been found. There is more detail following about whether this solution satisfies the termination criteria (set by options opter and optca).</td>
</tr>
<tr>
<td>9</td>
<td>INTERMEDIATE NON-INTEGER</td>
<td>An incomplete solution to a problem with discrete variables. A feasible solution has not yet been found. See section Model Termination Conditions for MIPs for more information.</td>
</tr>
<tr>
<td>10</td>
<td>INTEGER INFEASIBLE</td>
<td>It has been proven that there is no feasible solution to a problem with discrete variables. See section Model Termination Conditions for MIPs for more information.</td>
</tr>
<tr>
<td>11</td>
<td>LIC PROBLEM - NO SOLUTION</td>
<td>The solver cannot find the appropriate license key needed to use a specific subsolver.</td>
</tr>
<tr>
<td>12</td>
<td>ERROR UNKNOWN</td>
<td>After a solver error the model status is unknown.</td>
</tr>
<tr>
<td>13</td>
<td>ERROR NO SOLUTION</td>
<td>An error occurred and no solution has been returned. No solution will be returned to GAMS because of errors in the solution process.</td>
</tr>
<tr>
<td>14</td>
<td>NO SOLUTION RETURNED</td>
<td>A solution is not expected for this solve. For example, the CONVERT solver only reformats the model but does not give a solution.</td>
</tr>
<tr>
<td>15</td>
<td>SOLVED UNIQUE</td>
<td>Indicates the solution returned is unique, i.e. no other solution exists. Used for CNS models. Examples where this status could be returned include non-singular linear models, triangular models with constant non-zero elements on the diagonal, and triangular models where the functions are monotone in the variable on the diagonal.</td>
</tr>
<tr>
<td>16</td>
<td>SOLVED</td>
<td>Indicates the model has been solved: used for CNS models. The solution might or might not be unique. If the solver uses status 17 - SOLVED SINGULAR wherever possible then this status implies that the Jacobian is non-singular, i.e. that the solution is at least locally unique.</td>
</tr>
<tr>
<td>17</td>
<td>SOLVED SINGULAR</td>
<td>Indicates the CNS model has been solved, but the Jacobian is singular at the solution. This can indicate that other solutions exist, either along a line (for linear models) or a curve (for nonlinear models) including the solution returned.</td>
</tr>
<tr>
<td>18</td>
<td>UNBOUNDED - NO SOLUTION</td>
<td>The model is unbounded and no solution can be provided.</td>
</tr>
<tr>
<td>19</td>
<td>INFEASIBLE - NO SOLUTION</td>
<td>The model is infeasible and no solution can be provided.</td>
</tr>
</tbody>
</table>

4.38.6.1.1 Model Status  Note that the model status is stored in the model attribute `modelStat`. For details on model attributes, see section Model Attributes. Observe that there are compile-time constants
that are related to this model attribute.

<table>
<thead>
<tr>
<th>Value</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NORMAL COMPLETION</td>
<td>The solver terminated in a normal way: it was not interrupted by a limit (resource, iterations, nodes or other) or by internal difficulties. The model status describes the characteristics of the accompanying solution.</td>
</tr>
<tr>
<td>2</td>
<td>ITERATION INTERRUPT</td>
<td>The solver was interrupted because it used too many iterations. The option iterlim may be used to increase the iteration limit if everything seems normal.</td>
</tr>
<tr>
<td>3</td>
<td>RESOURCE INTERRUPT</td>
<td>The solver was interrupted because it used too much time. The option reslim may be used to increase the time limit if everything seems normal.</td>
</tr>
<tr>
<td>4</td>
<td>TERMINATED BY SOLVER</td>
<td>The solver encountered some difficulty and was unable to continue. More details will appear following the message.</td>
</tr>
<tr>
<td>5</td>
<td>EVALUATION INTERRUPT</td>
<td>Too many evaluations of nonlinear terms at undefined values. We recommend to use variable bounds to prevent forbidden operations, such as division by zero. The rows in which the errors occur are listed just before the solution.</td>
</tr>
<tr>
<td>6</td>
<td>CAPABILITY PROBLEMS</td>
<td>The solver does not have the capability required by the model. For example, some solvers do not support certain types of discrete variables or support a more limited set of functions than other solvers.</td>
</tr>
<tr>
<td>7</td>
<td>LICENSING PROBLEMS</td>
<td>The solver cannot find the appropriate license key needed to use a specific subsolver.</td>
</tr>
<tr>
<td>8</td>
<td>USER INTERRUPT</td>
<td>The user has sent a message to interrupt the solver via the interrupt button in the IDE or sending a Control+C from a command line.</td>
</tr>
<tr>
<td>9</td>
<td>ERROR SETUP FAILURE</td>
<td>The solver encountered a fatal failure during problem set-up time.</td>
</tr>
<tr>
<td>10</td>
<td>ERROR SOLVER FAILURE</td>
<td>The solver encountered a fatal error.</td>
</tr>
<tr>
<td>11</td>
<td>ERROR INTERNAL SOLVER FAILURE</td>
<td>The solver encountered an internal fatal error.</td>
</tr>
<tr>
<td>12</td>
<td>SOLVE PROCESSING SKIPPED</td>
<td>The entire solve step has been skipped. This happens if execution errors were encountered and the GAMS parameter ExecErr has been set to a nonzero value or the property MaxExecError has a nonzero value.</td>
</tr>
<tr>
<td>13</td>
<td>ERROR SYSTEM FAILURE</td>
<td>This indicates a completely unknown or unexpected error condition.</td>
</tr>
</tbody>
</table>

### 4.38.6.1.2 Solver Status

Note that the solver status is stored in the model attribute solveStat. For details on model attributes, see section **Model Attributes**. Observe that there are compile-time constants that are related to this model attribute.

### 4.38.6.2 Solver Report

The next section in the listing file is the part of the solve summary that is particular to the solver program that has been used. This section normally begins with a message identifying the solver and its authors: **MINOS and QUADMINOS** was used in the example here. There will also be diagnostic messages in plain language if anything unusual was detected and specific performance details, some of them probably
technical. The solver manual of the respective solver will help explain these. In case of serious trouble, the GAMS listing file will contain additional messages printed by the solver. This may help identify the cause of the difficulty. If the solver messages do not help, a perusal of the solver documentation or help from a more experienced user is recommended. The solver report from our Illustrative Model follows.

GAMS/MINOS
MINOS 5.6  (Nov 2014)

GAMS/Minos 5.6, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 0.77 Mb

EXIT - Optimal Solution found, objective: 2.899038

Note that the line Work space allocated -- 0.77 MB provides the amount of memory used by the solver for the problem. The solver estimates the amount of memory it will need to solve the problem. If this amount is not available on the machine used, GAMS will return a message saying that not enough memory was allocated. In addition, GAMS will return the maximum amount of memory available on the machine. The user may direct the solver to use less memory with the model attribute .workspace:

mymodel.workspace = x;

Here mymodel is the name of the model being solved as specified by the model statement and x is the amount of memory in Megabytes. Note that the solver will attempt to solve the problem with x MB of memory. However, it is not guaranteed to succeed since the problem may require more memory.

Note that more information for a successful run may be obtained using the option sysout. As usual, the respective option statement should be placed before the solve statement.

4.38.6.3 The Solution Listing

The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed with four pieces of information. The solution listing may be suppressed with the option solprint:

option solprint = off;

This option statement should be placed before the solve statement. The solution listing generated from our Illustrative Model is shown below.

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQU fsum</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>EQU dmean</td>
<td>10.0000</td>
<td>10.0000</td>
<td>10.0000</td>
</tr>
<tr>
<td>EQU dvar</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

fsum fractions must add to 1.0

dmean definition of mean return on portfolio
dvar definition of variance

---- VAR x fraction of portfolio invested in asset i

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardware</td>
<td>.</td>
<td>0.3029</td>
<td>+INF</td>
</tr>
<tr>
<td>software</td>
<td>.</td>
<td>0.0865</td>
<td>+INF</td>
</tr>
<tr>
<td>show-biz</td>
<td>.</td>
<td>0.5048</td>
<td>+INF</td>
</tr>
<tr>
<td>t-bills</td>
<td>.</td>
<td>0.1058</td>
<td>+INF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>variance</td>
<td>-INF</td>
<td>2.8990</td>
<td>+INF</td>
</tr>
</tbody>
</table>

variance variance of portfolio

The order of the equations and variables is the same as in the symbol listing map.

The four columns associated with each entry correspond to the equation and variable attributes and have the following meaning:

- **LOWER** lower bound (.lo)
- **LEVEL** level value (.l)
- **UPPER** upper bound (.up)
- **MARGINAL** marginal (.m)

For variables, the values in the **LOWER** and **UPPER** columns refer to the lower and upper bounds. For equations, they are obtained from the (constant) right-hand side value and from the relational type of the equation. For details see section Variable Attributes and Equation Attributes. Note that instead of level values, slack values may be shown in equations. For more information, see section Customizing the Output File below.

Note

The **LEVEL** and **MARGINAL** values have been determined by the solver and the values shown are used to update the GAMS values. In the list they are shown with fixed precision, but the values are returned to GAMS with full machine accuracy. The single dots '.' on the list represent zeros.

**EPS** is the GAMS extended value used for a stored zero. It is common to see a marginal value given as EPS, since GAMS uses the convention that marginals are necessarily zero for basic variables and typically non-zero for other variables.

Note

**EPS** is used to indicate non-basic variables whose marginal values are at or close to zero, and for nonlinear problems to indicate superbasic variables whose marginals are at or close to zero. A superbasic variable is one between its bounds at the final point but not in the basis.

Note that in the Glossary there are brief explanations of technical terms that were used in this section.

For models that are not solved to optimality or for models types without an objective, some constraints may additionally be marked with certain flags. The list of these flags and their description is given in the following table.
<table>
<thead>
<tr>
<th>Shorthand Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFES</td>
<td>The row or column is infeasible. This mark is made for any entry where the level value is not between the upper and lower bounds.</td>
</tr>
<tr>
<td>NOPT</td>
<td>The row or column is non-optimal. This mark is made for any non-basic entries for which the marginal sign is incorrect, or superbasic ones for which the marginal value is too large.</td>
</tr>
<tr>
<td>UNBND</td>
<td>The row or column can increase without limit (e.g. in an unbounded ray) or has an excessively large magnitude.</td>
</tr>
<tr>
<td>REDEF</td>
<td>The equation type has been ignored or redefined for this solution point, e.g. because this is an MCP</td>
</tr>
<tr>
<td>REDIR</td>
<td>The equation has been flipped (i.e. negated or reoriented) in the model statement: useful for MCP and EMP</td>
</tr>
<tr>
<td>DEPND</td>
<td>The row or column is dependent on other rows or columns in the system: see CNS</td>
</tr>
</tbody>
</table>

### 4.38.6.4 Report Summary

The final section of the solution report is the report summary, marked with four asterisks (as are all important components of the output) followed by the title REPORT SUMMARY. It shows the count of rows or columns that have been marked INFES, NOPT or UNBND in the solution listing. For model types like MCP where REDEFs and REDIRs are possible, the counts for these markers may also be shown. The sum of infeasibilities will be shown if the reported solution is infeasible. The domain error count is only shown if the problem is nonlinear. If there are variables or equations where the levels were projected to one of the bounds, the count of those is also shown here.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report   SOLVE portfolio Using NLP From line 50

**** REPORT SUMMARY :
  0    NONOPT
  0    INFEASIBLE
  0    UNBOUNDED
  0    ERRORS
 42    PROJECTED
```

### 4.38.7 Post-Solution Output

The final part of the listing file is the post-solution output. It contains Final Execution Summary and the File Summary.

#### 4.38.7.1 Final Execution Summary

The final execution summary is marked with the title Execution and contains the output from display statements that were placed after the solve statement in the model, allowing simple reporting. In addition, it shows the final execution time and memory use. The respective output from our Illustrive Model follows.
A Quadratic Programming Model for Portfolio Analysis ALAN, SEQ=124a)

**Execution**

---- 51 VARIABLE x.L fraction of portfolio invested in asset i

hardware 0.303, software 0.087, show-biz 0.505, t-bills 0.106

---- 51 VARIABLE variance.L = 2.899 variance of portfolio

EXECUTION TIME = 0.001 SECONDS 3 MB

For further information on this output and ways to customize it, see chapter The Display Statement.

### 4.38.7.2 File Summary

The file summary is the very last part of the output file. If output has been written to put files, there will be a REPORT FILE SUMMARY that is marked with four asterisks. The report will list the put files with their internal names and the full paths of their external names.

All listing files have a FILE SUMMARY that is marked with four asterisks and that reports the names of the input and output (listing) files.

**** FILE SUMMARY

<table>
<thead>
<tr>
<th>Input</th>
<th>C:\PROGRAM FILES\gamsIDE\ALAN.GMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>C:\PROGRAM FILES\gamsIDE\ALAN.LST</td>
</tr>
</tbody>
</table>

If work files (save or restart) have been used, they will be listed in the file summary as well.

### 4.38.8 Error Reporting

All comments and descriptions about errors have been collected in this section for easy reference when disaster strikes.

Effective error detection and recovery are important parts of any modeling system. GAMS is designed around the assumption that the error state is the normal state of modeling. Experience shows that most compilations during the early stages of development will produce errors. Not to worry! The computer is much better at checking details than the human mind and should be able to provide positive feedback and suggestions about how to correct errors or avoid ambiguities. Developing a model is like writing a paper or an essay; many drafts and rewrites are required until the arguments are presented in the most effective way and meet all the requirements of proper English. GAMS acts like a personal assistant with knowledge of mathematical modeling and of the syntactic and semantic details of the language.

Errors are detected at various stages in the modeling process. Most of them are caught at the compilation stage, which behaves like the proofreading stage of the modeling process. Once a problem has passed through the rigorous test of this stage, the error rate drops to almost zero. Most of the execution runs, which are much more expensive than compilation, proceed without difficulties because GAMS knows about modeling and has anticipated problems. Many of the typical errors made with conventional programming languages are associated with concepts that do not exist in GAMS. Those error sources – like address calculations, storage assignment, subroutine linkages, input-output and flow control – create problems at execution time, are difficult to locate, often lead to long and frustrating searches, and leave the computer user intimidated. GAMS takes a radically different approach. Errors are spotted as early as possible, they are reported in a way that is comprehensible to the user, including clear suggestions for how to correct the problem and a presentation of the source of the error in terms of the problem of the user.
Note

All errors are marked with four asterisks '****' at the beginning of a line in the output listing.

As soon as an error is detected, processing will be stopped at the next convenient opportunity. A model will never be solved after an error has been detected. The only remedy is to fix the error and repeat the run.

Errors are grouped into the three phases of GAMS modeling: compilation, execution and model generation (which includes the solution that follows). In the following subsections we will discuss the errors that may occur in each of these phases.

4.38.8.1 Compilation Errors

Compilation errors are discussed in some detail in the tutorial A GAMS Tutorial by Richard E. Rosenthal and the tutorial Fixing Compilation Errors. Note that there is some overlap between the material in the tutorials and this section. Several hundred different types of errors can be detected during compilation and can often be traced back to just one specific symbol in the GAMS input. Most of the errors are caused by simple mistakes: forgetting to declare an identifier, putting indices in the wrong order, leaving out a necessary semicolon or misspelling a label. For errors that are not caused by such simple mistakes, the explanatory error message text will help diagnose the problem and correct it.

Note

When a compilation error is discovered, a dollar symbol and error number are printed below the offending symbol (usually to the right) on a separate line that begins with four asterisks.

If more than one error is encountered on a line (possibly because the first error caused a series of other spurious errors) the dollar signs may be suppressed and error number squeezed. GAMS will not list more than 10 errors on any one line.

Note

A list of all error numbers encountered and a description of the probable cause of each error is printed at the end of the echo print of the program. The error messages are self-explanatory. For further information on compilation errors and advice on how to resolve them, see the tutorial Fixing Compilation Errors.

It is worth noting that it is easy to produce a model that does not do what you want it to do, but does not contain errors in the sense that the term is being used in this section. The best precaution is to check your work carefully and build in as many automatic consistency checks as possible.

Note that if a GAMS reserved word is accidentally used for the name of a label or an identifier it is impossible to provide helpful error messages for technical reasons. This may cause confusion. We recommend users to familiarize themselves with the reserved words.
Attention

In some cases an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions whose explanations seem quite silly. We recommend to always check carefully for the cause of the first error in such a group. If nothing obvious is wrong, look at the previous statement and particularly watch out for missing semicolons.

The following example illustrates the general reporting format for compilation errors.

1 Set c crops / wheat, corn, wheat, longaname /  
  **** $172
2 Parameter price(c) / wheat 200, cotton 700 /  
  **** $170
3
Error Messages

170 Domain violation for element  
172 Element is redefined  

**** 2 ERROR(S) 0 WARNING(S)  

**** USER ERROR(S) ENCOUNTERED

4.38.8.2 Compilation Time Errors

The reporting format for errors found while analyzing solve statements is more complicated than for normal compilation errors, mainly because many things must be checked. All identifiers referenced must be defined or assigned, the mathematics in the equations must match the model class, and so on. More elaborate reporting is required to accurately describe any problems found. The solve statement is only checked if the model has been found to be error free up to this point. This is not only because the check is comparatively expensive, but also because many erroneous and confusing messages can be produced while checking a solve in a program containing other errors.

Attention

Compiler error messages related to a solve statement are reported in two places and in two formats:

1. They are shown immediately after the solve statement with a short text including the name of any offending identifier and the type of model involved. This will be sufficient in most cases.
2. A longer message with some hints appears with the other error messages at the end of the compilation.

The example below illustrates the general reporting format for compiler errors associated with a solve statement.

1 Variables x, y, z ;  
2 Equations eq1, eq2 ;  
3  
4 eq1.. x**2 - y =e= z ;  
5 eq2.. min(x,y) =l= 20 ;  
6  
7 Model silly / all / ;  
8 solve silly using lp maximizing z ;  
  **** $54,51,256
THE FOLLOWING LP ERRORS WERE DETECTED IN MODEL SILLY:

54 IN EQUATION EQ1 .. ENDOG OPERANDS FOR **
51 IN EQUATION EQ2 .. ENDOG ARGUMENT(S) IN FUNCTION

Error Messages

51 Endogenous function argument(s) not allowed in linear models
54 Endogenous operands for ** not allowed in linear models
256 Error(s) in analyzing solve statement. More detail appears
   Below the solve statement above

3 ERROR(S) 0 WARNING(S)
USER ERROR(S) ENCOUNTERED

4.38.8.3 Execution Errors

Execution time errors are usually caused by illegal arithmetic operations such as division by zero
or taking the log of a negative number. GAMS prints a message on the output file with the line
number of the offending statement and continues execution. A GAMS program should never abort
with an unintelligible message from the computer's operating system if an invalid operation is
attempted. GAMS has rigorously defined an extended algebra that contains all operations including illegal
ones. Note that the model [CRAZY] contains all non-standard operations and should be executed
to study for its exceptions. For advice on detecting and resolving execution errors, see the tutorial
Finding and Fixing Execution Errors and Performance Problems.

Recall that the GAMS arithmetic is defined over the closed interval [−INF, +INF] and contains the
extended range arithmetic values EPS (small but not zero), NA (not available) and UNDF (the result of
an illegal operation). The results of illegal operations are propagated through the entire system and can
be displayed with standard display statements. However, observe that, if errors have been previously
detected, a model cannot be solved and a work file cannot be saved.

4.38.8.4 Solve Errors

The execution of a solve statement can trigger additional errors called matrix errors. They report on
problems encountered during transformation of the model into a format required by the solver. Problems
are most often caused by illegal or inconsistent bounds, or by an extended range value that is used as a
matrix coefficient. The example below shows the general format of these errors:

```
1 Variable x;
2 Equation eq1;
3
4 eq1.. x =l= 10 ;
5 x.lo = 10 ;
6 x.up = 5 ;
7 Model wrong /eq1/;
8 solve wrong using lp maximizing x ;
9
**** Matrix error - lower bound > upper bound
x (.LO, .L, .UP = 10, 0, 5)
...
**** SOLVE from line 8 ABORTED, EXECERROR = 1
...
**** USER ERROR(S) ENCOUNTERED
```
Some solve statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the solution report. Note that by default the subsystems will interrupt the solution process if arithmetic exceptions are encountered. This may be changed with the option domlim. They are then reported on the listing as shown in the following example:

```
Variable x, y;
Equation one;

one.. y =e= sqrt(10/x);
x.l = 10;
x.lo = 0;

Model divide / all /;
solve divide maximizing y using nlp;
```

**SOLVE SUMMARY**

MODEL divide
TYPE NLP
DIRECTION MAXIMIZE
SOLVER MINOS

**** SOLVER STATUS 5 Evaluation Interrupt
**** MODEL STATUS 7 Feasible Solution
**** OBJECTIVE VALUE 3.1622776602E+0149

RESOURCE USAGE, LIMIT 0.182 1000.000
ITERATION COUNT, LIMIT 0 2000000000
EVALUATION ERRORS 2 0

EXIT - Function evaluation error limit exceeded.

**** ERRORS/WARNINGS IN EQUATION one
2 error(s): div: FUNC SINGULAR: x/y, |y| <= 1e-150 (RETURNED 1E299)

**** REPORT SUMMARY : 1 NONOPT ( NOPT)
0 INFEASIBLE
0 UNBOUNDED
1 ERRORS ( ****)

Note that the solver status returned with a value of 5, meaning that the solver has been interrupted because more than domlim evaluation errors have been encountered. The type of evaluation error and the equation causing the error are also reported.

In case the solver returns an intermediate feasible solution because of evaluation errors, the following solve will still be attempted. The only fatal GAMS error that can be caused by a solver program, is the failure to return any solution at all. If this happens all possible information is listed on the GAMS output file and any solves that follow will not be attempted.

### 4.38.9 Customizing the Output File

This section reviews the most commonly used dollar control options, options, and command line parameters to customize output in the listing file. Table 1 lists dollar control options, options, and command line parameters that are used in the input file to control the amount of detail in the output...
file produced by the GAMS compiler. Table 2 lists dollar control options and command line parameters that can change the layout and appearance of the output. The first column of the two tables shows the purpose of customizing the output that differs from the GAMS default behavior.

The output generated by display statements can also be customized. This topic is covered in section Display Controls from chapter The Display Statement. See also the complete list of dollar control options that affect the output format in section Dollar Control Options Affecting the Output Format.

**Table 1: Customization of output to be included in the listing file**

<table>
<thead>
<tr>
<th>Customization</th>
<th>Method</th>
<th>Further Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>suppress the echo print</td>
<td>$offlisting</td>
<td>$onlisting restores the default behavior. Any lines between $offlisting and $onlisting will not appear in the echo print.</td>
</tr>
<tr>
<td>suppress include files in the echo print</td>
<td>$offinclude</td>
<td>$oninclude restores the default behavior.</td>
</tr>
<tr>
<td>activate the symbol reference map</td>
<td>$onSymXRef</td>
<td>Maps are most often turned on or off at the beginning of the program and left as initially set. $onSymXRef activates the symbol reference map and $offSymXRef restores the default behavior.</td>
</tr>
<tr>
<td>activate the symbol listing map</td>
<td>$onSymList</td>
<td>Maps are most often turned on or off at the beginning of the program and left as initially set. $onSymList activates the symbol listing map and $offSymList restores the default behavior.</td>
</tr>
<tr>
<td>activate the unique element listing</td>
<td>$onUELList</td>
<td>Maps are most often turned on or off at the beginning of the program and left as initially set. $onUELList activates the unique element listing map and $offUELList restores the default behavior.</td>
</tr>
<tr>
<td>activate the unique element reference map</td>
<td>$onUELXRef</td>
<td>Maps are most often turned on or off at the beginning of the program and left as initially set. $onUELXRef activates the unique reference map $offUELXRef restores the default behavior.</td>
</tr>
<tr>
<td>suppress or expand the equation listing</td>
<td>option limrow</td>
<td>The statement option limrow = 0; will suppress the equation listing; the statement option limrow = n; will expand it to n equations. The statement must be placed before The Solve Statement.</td>
</tr>
<tr>
<td>suppress or expand the column listing</td>
<td>option limcol</td>
<td>The statement option limcol = 0; will suppress the column listing; the statement option limcol = n; will expand it to n columns. The statement must be placed before The Solve Statement.</td>
</tr>
<tr>
<td>suppress the solution listing</td>
<td>option solprint</td>
<td>The statement option solprint = off; will suppress the solution listing. The statement must be placed before The Solve Statement.</td>
</tr>
</tbody>
</table>
customization | method | further details
--- | --- | ---
restrict output to just a few displays | save restart feature | with this strategy the listing file will contain only the output generated by the desired display statements. this facilitates concentrating on a narrow set of output while remaining capable of generating a lot more output. for details see section generating concise listing files in chapter the save and restart feature.
show slack variables in the solution listing | option solslack | the statement option solslack = 1; causes the equation output in the solution listing to show slack (.slack) variable values instead of level (.l) values. the statement must be placed before the solve statement.

<table>
<thead>
<tr>
<th>customization</th>
<th>further details</th>
</tr>
</thead>
<tbody>
<tr>
<td>change the default header</td>
<td>the directive $title causes every page on the output to have the header specified in text. the header may be reset later by using another line starting with $title. currently, the text may have up to 80 characters.</td>
</tr>
<tr>
<td>add a subheader</td>
<td>the statement $stitle text causes every page on the output to have the subheader specified in text. the header may be reset later by using another line starting with $stitle. currently, the text may have up to 80 characters.</td>
</tr>
<tr>
<td>start a new page in the echo print</td>
<td>the directive is $eject, where n is the number of lines left on the current page.</td>
</tr>
<tr>
<td>start a new page in LST file if less than n lines are left</td>
<td>the general syntax is: gams mymodel pw=n ps=m. here n is desired page width and the desired page length is m. note that pw and ps are synonyms of pageWidth and pageSize respectively.</td>
</tr>
<tr>
<td>change width and length of a page</td>
<td>the directive $onUpper restores the default.</td>
</tr>
<tr>
<td>change the echo print to upper case only</td>
<td>$onUpper</td>
</tr>
<tr>
<td>change lines in the echo print to be double spaced</td>
<td>$double</td>
</tr>
<tr>
<td>change the marker ****</td>
<td>$stars</td>
</tr>
</tbody>
</table>
4.39 GAMS Log

4.39.1 Introduction

The GAMS log provides the log of a GAMS run including the GAMS job description, defined command
line parameters, license information, solver log, and GAMS compilation and execution output. Looking at
the log gives a brief idea about the success of a GAMS run and is usually a useful first step in debugging
a GAMS model. The detailed output from GAMS is displayed in the output file. Log output can be
obtained in the form of a text file (typically with .log extension) that is generated with each GAMS run
based on the command line parameters logOption and logFile. The GAMS log consists of three major
sections: header, compilation log, and execution log.

4.39.2 Header

The header of the GAMS log consists of information that is independent of the content of the executed
GAMS file. It contains the job description, parameter log, and license log.

4.39.2.1 Job Description

In this line, the name of the GAMS file being run along with a timestamp, the GAMS version and
operating system information is provided.

--- Job trnsport.gms Start 08/30/21 17:29:15 36.1.0 r2c0a44a WEX-WEI x86 64bit/MS Windows

4.39.2.2 Parameter Log

The default parameters for the operating system and configuration are obtained via parameter file
(gmsprmNT.txt on Windows, gmsprmun.txt on Linux, gmsprmun.txt on MacOS) and gamsconfig.yaml
files. The parameter log explicitly lists all additional parameters in the following lines. For example, input
displays the full path of the GAMS input file. In the example log provided below, the output file option
PageSize is set to 0. ScrDir is the full path to the scratch directory; SysDir is the full path to the systems
directory, logOption is defined that controls the location of the output log.

--- Applying:
C:\GAMS\36\gmsprmNT.txt
C:\Users\jsmith\Documents\GAMS\gamsconfig.yaml

--- GAMS Parameters defined
Input C:\test\trnsport.gms
PageSize 0
ScrDir C:\test\225a\nSysDir C:\GAMS\36\nLogOption 3
ErrMsg 1
ErrorLog 99
IDE 1
LstTitleLeftAligned 1
4.39.2.3 License Log

This section provides information about the license such as licensee, the organization, and the license number is provided on the second line. The full path to the license file that is being read is provided on the third line. The last line in the image shows the GAMS license type being used.

Licensee: John Smith
The ACME Corporation, USA
C:\Users\jsmith\Documents\GAMS\gamslice.txt
Other time limited license

4.39.2.4 Processor Information and GAMS version

This section provides information about the processor and GAMS version being used.

Processor information: 1 socket(s), 8 core(s), and 16 thread(s) available
GAMS 36.1.0 Copyright (C) 1987-2021 GAMS Development. All rights reserved

4.39.3 Compilation Log

A GAMS run consists of two phases: compilation and execution. The following lines provide details about the compilation, memory usage, the time elapsed in compilation. The number in parentheses refers to the line number in the echo print of the input file. In addition to the above-mentioned output, this part of the log also contains information about the output such as GDX files, the files included using $include, and outputs of commands such as $call as shown in the following lines.

--- Starting compilation
--- clad.gms(27) 2 Mb
--- $echo File C:\test\includefilename.gms
--- clad.gms(30) 2 Mb
--- . \includefilename.gms(1) 2 Mb
--- clad.gms(32) 2 Mb
--- call echo "hello!"
"hello!"
--- clad.gms(45) 3 Mb
--- GDIn=C:\test\claddat.gdx
--- GDX File ($gdIn) C:\test\claddat.gdx
--- clad.gms(164) 3 Mb

4.39.4 Execution Output

The beginning of the execution phase of the GAMS run is clearly indicated in the GAMS log:

--- Starting execution: elapsed 0:00:00.010

In general, the execution log output indicates the progress of the GAMS job, including line numbers, memory usage, elapsed time for the run, and time required for long-running statements. Several statements also result in specific log output.

Output of GDX files generated using execute_unload is shown in the log as follows.
--- GDX File (execute_unload) path\to\outputfile.gdx

The output for reference files generated is shown in the log as follows.

--- RefFile path\to\trnsport.ref

Similarly, the information about a file written using the Put writing facility is mentioned in the log as follows.

--- Putfile results path\to\results.txt

The output of the program flow control features such as abort is shown as follows.

*** Error at line 61: Execution halted: abort$1 'model not solved as expected'

If there are loops in the program, the log shows information about the current iteration as follows.

--- LOOPS scen = s1

Similar to the output of call command shown during compilation phase, the output for execute command is shown in the execution phase. The following line of GAMS code

```gams
eexecute "echo output of execute command";
```

will result in the following output in the log.

--- trnsport.gms(67) 4 Mb
output of execute command

For each solve statement, there will also be a solver log embedded in the execution output (see below for details regarding the solver log). After a solver log, the execution output continues. It ends with the status of the GAMS Job (here *** Status: Normal completion), a timestamp and the total run time of the GAMS job.

--- Executing after solve: elapsed 0:00:02.892
--- trnsport.gms(66) 4 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 08/30/21 17:29:18 elapsed 0:00:02.892
4.39.4.1 Solver Log

One of the most important GAMS statements is the solve statement: a solve results in a substantial amount of log output.

The first line in the execution output is about model generation. Model generation can take a long time in which case the log also shows progress and equations that take long to generate.

--- Generating LP model transport
--- transport.gms(64) 4 Mb
--- 6 rows 7 columns 19 non-zeroes

This is followed by some statistics about the range of values seen in the model:

--- Range statistics (absolute non-zero finite values)
--- RHS [min, max] : [ 2.750E+02, 6.000E+02] - Zero values observed as well
--- Bound [min, max] : [ NA, NA] - Zero values observed as well
--- Matrix [min, max] : [ 1.260E-01, 1.000E+00]

After model generation, the solver name and solveLink are displayed.

--- Executing CPLEX (Solvelink=2): elapsed 0:00:02.066

The log output following this line is specific to the solver and will usually display the objective function value.

Optimal solution found
Objective: 153.675000

4.40 The GAMS Call and Command Line Parameters

There are multiple ways to trigger the run of a GAMS model: Running a model via F9 from GAMS Studio, executing GAMSJob.Run method in the object oriented APIs or calling the gams executable from a command line. In all cases the same GAMS engine runs the user’s model. Several options are available to customize the GAMS run. Depending on the particular way GAMS is triggered these options are supplied in different ways. For example, in Studio via the parameter editor in the toolbar, in the object oriented API through the GAMSOptions class, and from the command line via command line arguments. In order to avoid confusion with the often used word option we refer to theses entities as command line parameters and demonstrate its with the gams executable from the command line. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

In this chapter we will introduce how GAMS is called from the command line and how parameters may be specified on the command line or in other ways. In addition, we will discuss user-defined command line parameters, compile-time variables and compile-time constants, which are GAMS specialties, and environment variables in GAMS. Moreover, most of the chapter is dedicated to the detailed description of all GAMS options (i.e. options available via command line parameters, option statement, and model attribute.

In almost all cases such a run continues from start to end without any user interaction. A GAMS run usually consists of two phases: the compilation and execution phase.
4.40.1 The Generic GAMS Call

The simplest way to start GAMS from a command shell is to enter the following command from the system prompt:

> gams myfile

GAMS will compile and execute the GAMS statements in the file myfile. If a file with this name cannot be found, GAMS will look for a file with the extended name myfile.gms. During the run GAMS will print a log to the console and create a listing file that is written by default to the file myfile.lst. For example, the following statements retrieves and runs the model [TRNSPORT] from the GAMS model library with the responds from the GAMS system:

> gamslib trnsport
Copy ASCII : trnsport.gms
> gams trnsport
--- Job trnsport Start 06/21/17 06:23:45 24.8.4 r60966 WEX-WEI x86 64bit/MS Windows
GAMS 24.8.4 Copyright (C) 1987-2017 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC 0B71201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CBC: elapsed 0:00:00.018

COIN-OR CBC 24.8.4 r60966 Released Apr 10, 2017 WEI x86 64bit/MS Windows

COIN-OR Branch and Cut (CBC Library 2.9)
written by J. Forrest

Calling CBC main solution routine...
0 Obj 0 Primal inf 900 (3)
4 Obj 153.675
Optimal - objective value 153.675
Optimal objective 153.675 - 4 iterations time 0.052

Solved to optimality.
--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/21/17 06:23:46 elapsed 0:00:01.249

Note
GAMS can also run source files that have been compressed with Posix utility gzip or have been encrypted with tool ENDECRYPT, for example

> gamslib trnsport
> gzip trnsport.gms
> gams trnsport.gms.gz
results in

- retrieval of the [TRNSPORT] model from the GAMS Model Library
- compression the source file transport.gms
- GAMS running the compressed file transport.gms.gz.

Observe that some GAMS options may be specified on the command line as part of the GAMS call. In addition, command line parameters may be set by specifying a secondary customization parameter file and by modifying the GAMS system parameter file. As command line parameters may be specified in several different ways, there are rules of precedence in case of conflicting instructions. We will discuss these topics in the next four subsections.

### 4.40.1.1 Specifying Options Through the Command Line

GAMS permits certain options to be passed through the command line. The syntax of the simple GAMS call is extended as follows:

\[
> \text{gams myfile key1=value1 key2=value2 ...}
\]

Here `key1` is the name of the option that is being set on the command line and `value1` is the value to which the option is set. Depending on the option, `value1` could be a character string, an integer number or a real number. Note that the options that may be set on the command line are called *GAMS command line parameters*.

For example, consider the following commands to run the transportation model [TRNSPORT]:

\[
> \text{gams transport o myrun.lst logOption 2}
> \text{gams transport -o myrun.lst -logOption 2}
> \text{gams transport /o myrun.lst /logOption 2}
> \text{gams transport o=myrun.lst logOption=2}
> \text{gams transport -o=myrun.lst -logOption=2}
> \text{gams transport /o=myrun.lst /logOption=2}
\]

All six commands above are equivalent: each directs the output listing to the file `myrun.lst`. Note that `o` is the synonym of the command line parameter `output` and it is set to the value `myrun.lst`. In addition, the parameter `LogOption` is set to 2, which has the effect that the log output is redirected to the file `transport.log`. Please also note that the option name can be specified without consideration of the casing of the option. Hence `logoption` works as well as `LogOption` or any other casing. The way options are specified can vary with each option. So a mixed alternative, e.g. `gams transport -o=myrun.lst /logoption=3` is valid too.

In addition to predefined command line parameters, GAMS also allows user-defined command line parameters which work in tandem with compile-time variables. User-defined parameters, also called *double dash parameters*, and compile-time variables are introduced in section Double Dash Parameters, Compile-Time Variables and Environment Variables below.
4.40.1.1 Option Values with Spaces  

One may want to set string valued options including double dash parameters to an empty string or a string containing spaces, even trailing or leading spaces. All this requires the use of quotes. Quotes may be handled differently, depending on the operating system. Users should be attentive when using such string values. This is also important if GAMS is called from within a GAMS program with $call [=]gams ... and execute [=]gams ... because platform independence might be violated.

A platform independent way to set a string value with spaces is to use the variant that separates the option name and option value by a space, e.g. opt "". The variant without space, e.g. opt="" might cause issues on some Unix shells. Consider the following example where spacy.gms contains only the following line to write the content of the double dash parameter dd to the log:

$log value of dd: >%dd%<

Here is a sequence of GAMS runs with a variety of string values

> gams spacy --dd normal
> gams spacy --dd "normal"
> gams spacy --dd ""
> gams spacy --dd " leading"
> gams spacy --dd "trailing "
> gams spacy --dd " leading and trailing "
> gams spacy --dd " "

that result on all platforms in the following log output (only relevant part):

value of dd: >normal<
value of dd: >normal<
value of dd: ><
value of dd: > leading<
value of dd: >trailing <
value of dd: > leading and trailing <
value of dd: > <

An even better way to pass arguments to a GAMS run without the system shell interfering with interpretation of some characters is described in the next section.

Attention

GAMS options specifying file names may also contain leading and trailing blanks. Please be aware that while leading blanks in a file name are significant on all platforms, trailing blanks are only significant on some platforms (e.g. Linux). Hence output "mymodel.lst " and output "mymodel.lst" refer to the same file on Windows and to different files on Linux.

4.40.1.2 Specifying Options Through a Secondary Parameter File

Command line parameters may also be set by specifying a secondary customization parameter file. For example, we will create a file with the following two lines, call it moreOptions.txt and save it in the current working directory.

limCol=0
limRow=0
Note that setting \texttt{limcol} and \texttt{limrow} to zero will suppress the column listing and equation listing respectively. Moreover, note that we can specify the option in different way in this secondary parameter file, even have multiple options on one line. In the next step we will use the following command to run the transportation model [\texttt{TRNSPORT}]:

\begin{verbatim}
> gams trnsport parmFile=moreOptions.txt
\end{verbatim}

Note that this call has the same effect as the following specification:

\begin{verbatim}
> gams trnsport limCol=0 limRow=0
\end{verbatim}

Observe that command line parameter include files are particularly useful if modelers wish to use the same set of command line parameters repeatedly.

If options are listed multiple times on the command line, the last specification sets the option. This is important in particular in combination with a secondary parameter file. In the following example, GAMS will operate with \texttt{limRow=0}:

\begin{verbatim}
> gams trnsport limRow=10 parmFile=moreOptions.txt
\end{verbatim}

while with the order reversed, GAMS operates with \texttt{limrow=10}:

\begin{verbatim}
> gams trnsport parmFile=moreOptions.txt limRow=10
\end{verbatim}

\subsection*{4.40.1.3 Specifying Options Through the GAMS System Parameter File}

A third way to specify command line parameters is by modifying the GAMS system parameter file, which is part of the GAMS system. It has different names depending on the operating system: \texttt{gmsprmnt.txt} for Windows and \texttt{gmsprmun.txt} for UNIX/Linux and Mac OS X. The parameter file may be modified in the following way:

\begin{verbatim}
P*******************************
* GAMS 2.50 default Unix parameter file  *
* Gams Development Corp.            *
* Date : 4 May, 1998               *
*******************************
entries required by CMEX, put in by the gams script
SYSDIR
SCRDIR
SCRIPTNEXT
INPUT
PageWidth 95
ParmFile "c:\some\central\location\moreOptions.txt"
\end{verbatim}

Note that the last two lines were added to the standard GAMS system parameter file. As a consequence, each GAMS run will have a print width of 95 columns on the pages of the listing file. In addition, the command line parameters specified in the file \texttt{moreOptions.txt} will now apply to each GAMS run.
4.40.1.4 The GAMS Configuration in YAML Format

The GAMS configuration file `gamsconfig.yaml` written in the YAML format is easy to process by humans and computers. The GAMS Configuration Editor can be used to view and edit the GAMS configuration file but any other text editor can be used as well. This file has three configuration sections: command line parameters (`commandLineParameters`), operating system environment variables (`environmentVariables`), and external solver configuration (`solverConfig`). The latter is only relevant if you need to make a solver available to GAMS that is not shipped with the GAMS distribution. The COIN-OR GAMSlinks project has solver (links), e.g. Couenne, that could be added this way.

The first two sections are most easily described by an example:

```
commandLineParameters:
  - intVarUp:
    value: 1
  - eolCom:
    value: '//'
    maxVersion: 31
  - eolCom:
    value: '#'
    minVersion: 32
environmentVariables:
  - GDXCOMPRESS:
    value: 1
    maxVersion: 31
  - GDXCOMPRESS:
    value: 0
    minVersion: 31.1.0
    maxVersion: 31.1.0
    pathVariable: True
  - PATH:
    value: /my/path/to/R/bin:/my/path/to/R/site-bin
    minVersion: 31.1.0
    maxVersion: 31.1.0
```

In this example, the `gamsconfig.yaml` initializes some command line parameters and some environment variables. It initializes `intVarUp` unconditionally to 1. It also sets `eolCom` to the character sequence `//` for GAMS version 31 and earlier, while it uses the character `#` for GAMS version 32 and later. The keywords `minVersion` and `maxVersion` allow to have different entries for the same command line parameter for different versions. Also the file `gamsconfig.yaml` allows the repetition of entries. Hence, when GAMS runs it will filter all entries with appropriate `minVersion` and `maxVersion` and applies the setting of entries in sequence, so the last entry wins. Similarly, GAMS sets the environment variables `GDXCOMPRESS` to 0 or 1 depending on the actual version of GAMS. The last entry to set the `PATH` environment variable will be only applied for the GAMS version 31.1.0. Moreover, the indicator `pathVariable` set to `True` will result in prefixing the environment variable `PATH` with `/my/path/to/R/bin:/my/path/to/R/site-bin`.

The YAML syntax for the `gamsconfig.yaml` file is straightforward. Each section starts with a fixed name (`commandLineParameters` or `environmentVariables`) and is followed by a list of entries. Each entry starts with `-` and the name of the command line parameter or the name of the environment variable followed by a colon `:`. The next lines start with a keyword (`value`, `minVersion`, `maxVersion` and for environment variables also `pathVariable`) followed by a colon `:` and a value. The syntax of YAML files is very simple but it is crucial to use a consistent alignment. So all entries of a section must be perfectly aligned. Please be careful when mixing spaces and the tab character. If GAMS complains during the processing of a `gamsconfig.yaml` file with e.g. `System problems in yaml_parser_parse: mapping values are not allowed in this context` most likely your alignment is off as in the following erroneous example:
commandLineParameters:
  - eolCom:
    value: '//'
    maxVersion: 31

The lines with `value` and `maxVersion` need to start in the same column. There are plenty of online syntax checkers for YAML files. Use your favorite search engine and search for `yaml validator` (check the popular YAML validator) and copy and paste your problematic `gamsconfig.yaml` to get more insight into the YAML syntax problem. Moreover, the `gamsconfig.yaml` file has some required fields and some requirements for the value of some keys. In fact there is a schema file that validates that a syntactically correct `gamsconfig.yaml` fulfills the GAMS schema for configuration files. The schema file is part of the distribution and can be found in the system directory (`gamsconfig_schema.json`). One can validate a `gamsconfig.yaml` against this schema via a Python program (you might need to install additional Python packages `jsonschema` and `ruamel.yaml`):

```python
import jsonschema
from ruamel.yaml import YAML
import json

with open('gamsconfig_schema.json') as f:
    schema = json.load(f)

yaml = YAML(typ="safe")
with open('gamsconfig.yaml') as f:
    config = yaml.load(f)

print(jsonschema.validate(instance=config, schema=schema))
```

Besides the required fields like `value` the schema also imposes restrictions on some value types (e.g. `True` and `False` for `pathVariable` and a proper release number for `minVersion` and `maxVersion`). The form of the release number can be just the major release (e.g. 31), the major and minor release number (e.g. 31.1) or the full detailed release number, e.g. 31.1.0. So if `maxVersion` is set to e.g. 31 then all releases of 31 fulfill this version requirement.

Some parts of the GAMS system react on environment variables, e.g. the environment variable `GMSPYTHONLIB` decides about the Python library and installation used for embedded Python code. At the start of a GAMS run environment variables are set to the values given in the `gamsconfig.yaml` file. This is a convenient way to manage GAMS-related environment variables without utilizing OS-specific ways to set environment variables.

GAMS will process a sequence of `gamsconfig.yaml` files. It starts with the file in the GAMS system directory and then searches through some system-wide and user-specific standard locations.

Note

With GAMS 32 new defaults for several options were introduced. We have collected these options and their old or legacy default values in the file `gamsLegacyConfig.yaml` located in the GAMS system directory. Copy this file as `gamsconfig.yaml` to a location searched by GAMS to continue to work with the legacy defaults. The `minVersion` keyword in this file indicates when the new default became active.
4.40.1.4.1 Adding Solvers via gamsconfig.yaml

The third section in the GAMS configuration file gamsconfig.yaml allows to make a solver not shipped with GAMS available to the GAMS system. For that, an interface between an instantiation of a GAMS model and a solver needs to be implemented by use of libraries GMO and GEV. The COIN-OR GAMSlinks project provides examples of such solver interfaces. For the remainder of this section assume that we want to introduce a new version of the MINLP/MIQCP solver SHOT. GAMS knows about available solvers from the sub-system configuration file gmscmpNT.txt (gmscmpun.txt for non-Windows platforms). Here is a section from this file for the MINLP/MIQCP solver SHOT on Windows:

```
SHOT 1001 5 000102030405 1 0 2 MINLP MIQCP

```

The gamsconfig.yaml file provides the same information in a more descriptive way:

```
solverConfig:
  - SHOT2:
      minVersion: 33
      fileType: 1001
      dictType: 5
      licCodes: 000102030405
      defName: optshot.def
      scriptName: gmsgennt.cmd
      executableName: gmsgennx.exe
      library:
        libName: D:\Users\janeDoe\Downloads\gamslink\GamsShot-20.dll
        auditCode: sht
        solverInterfaceType: 1
        threadSafeIndic: True
      modelTypes:
        - MINLP
        - MIQCP

```

The new solver will be called SHOT2 and has almost the identical information as the solver SHOT that comes with GAMS. We could also use the name SHOT again. This would hide the original solver SHOT and use the one described by gamsconfig.yaml. In addition to the solver information, this section allows the keywords min/maxVersion that instructs whether to include the solver depending on the GAMS version.

The `fileType` (line 1 second item in the section of gmscmpNT.txt) determines the type of the scratch files that contain the model instance. The one-digit determines binary (1) or text (0) format of the scratch files. The ten-digit determines whether GAMS puts the initial marginal information into the scratch files (1 for writing, 0 for not writing). The scratch file with the constraint matrix is stored column wise, the hundred-digit instructs GAMS to write (if at 1) the row count per column, so the solver can layout memory in advance for storing the constraint matrix. With GAMS’ current libraries gmo and gev this information is not used and hence the hundred-digit should be at 0. The thousand-digit is about scaling. A non-zero value instructs GAMS to write the information stored in the .scale/.prior/.stage variable and equation attributes into the scratch files. With a value of 1 this is only done if scaleOpt/priorOpt are on. With a value of 2 this information is written independently of scaleOpt/priorOpt. With a value of 3 for models with variable and matching information (e.g. MCP) the reported marginals will be scaled by GAMS. The ten thousand-digit is always 0 and the hundred thousand-digit (if set to 1) instructs GAMS to write the variable/equation matching information to the scratch files for models that need this information (e.g. MCP). The leading zeros of the value for key `fileType` should be omitted.

The `dictType` (line 1 third item in the section of gmscmpNT.txt) decides about the type of the dictionary scratch file that contains the mapping information between the GAMS names and the variable and equation indexes seen by the solver. For example, this helps the solver to form good variable and equation names for error messages. Four values are allowed:
4.40 The GAMS Call and Command Line Parameters

- 0: no dictionary information
- 5: dictionary with all labels that are used in the model instance
- 15: dictionary with all labels that are used in GAMS program
- 25: dictionary with all labels that are used in GAMS model plus an extra GDX file with the entire GAMS database (similar to `execute unload "everything";` prior to the solve statement)

The **licCode** (line 1 fourth item in the section of `gmscmpNT.txt`) gives a guess about what license codes are required to run the solver. This is not the actual license information but allows other programs like e.g. "GAMS Studio" to list the solvers available with a particular license. In most cases solvers that are added this way should list the GAMS BASE license codes "000102030405".

The **defName** (line 2 second item in the section of `gmscmpNT.txt`) points to a file that contains all the solver options for the solver. This key does not need to be provided (same in `gmscmpNT.txt`) if the **defName** is `opt + solvername + .def` and the file resides in the GAMS system directory. Since we use the name SHOT2 GAMS would expect the solver options to be in `optshot2.def`. If there were an `optshot.def` in the GAMS system directory for the original SHOT, we could reuse that file and list `optshot.def` for the **defName** key. The **defName** provided here is used by other programs like "GAMS Studio" that can assist with building solver options files.

The next three items on line 1 in the section of `gmscmpNT.txt` are for internal purposes and not discussed here in detail. They can be set via `gamsconfig.yaml` via the keys `defaultOkFlag` and `hiddenFlag` with values True or False.

GAMS has different ways of calling a solver controlled by the **solveLink** option. The **scriptName** (line 2 in the section of `gmscmpNT.txt`) contains the name of the script that calls the solver (in case of `solveLink=0` (chainScript) or 1 (callScript)). For solvers that are capable of `solveLink=5` (loadLibrary) a generic script `gmsgennt.cmd` (or `gmsgenus.run`) can be used. The **executableName** (line 3 in the section of `gmscmpNT.txt`) provides the name of the solver executable. Again for solvers capable of `solveLink=5` (loadLibrary) a generic executable `gmsgennx.exe` (or `gmsgenux.out`) can be used.

The section library is necessary only for solvers capable of `solveLink=5` (loadLibrary). The **libName** (line 3 first item in the section of `gmscmpNT.txt`) contains the name of the solver library file (line 3 first item in the section of `gmscmpNT.txt`). The **auditCode** (line 3 second item in the section of `gmscmpNT.txt`) is the prefix of the functions in the solver library (e.g. `shtCallSolver`). The **solverInterfaceType** (line 3 third item in the section of `gmscmpNT.txt`) defines the API implemented by the solver library and loaded by GAMS. Currently, there are only two types: 1 (xyzReadyApi and xyzCallSolver) and 2 (xyzCallSolver only). The **threadSafeIndic** (line 3 third item in the section of `gmscmpNT.txt`) indicates whether the solver library is thread-safe and can be used in combination with `solveLink=6` (aSyncThreads).

The final section **modelTypes** is about the model types the solver can handle. This information can be found in line 1 after item 7 in `gmscmpNT.txt`.

### 4.40.1.5 Order of Precedence for Options

The order of precedence for command line parameters including customization specifications in the GAMS IDE is as follows:

1. Command line parameters that are specified on the command line or in the Studio/IDE command line field.
2. Command line specifications in the IDE options window **Execute** (this is reached through File|Options|Execute).
3. Command line specifications in the specific IDE dialogs, e.g. Output (this is reached through File|Options|...).

4. Entries in the GAMS configuration files gamsconfig.yaml.

5. Entries in the GAMS system parameter file.

Many command line parameters initialize the default for a GAMS Option also accessible inside the GAMS program. For example, the system default for option MIQCP is SBB. If the command line parameter MIQCP has been set, e.g. to DICOPT, DICOPT will be new default MIQCP solver. Inside the GAMS program, one can reset the MIQCP via option MIQCP=SHOT; but setting it back to the default will result in DICOPT (not SBB) being the default solver: option MIQCP=default;.

### 4.40.2 Double Dash Parameters, Compile-Time Variables and Environment Variables

In this section we will cover double-dash parameters that enable users to set values for specific variables on the command line. These variables are substituted with their respective specified values at compile time. We will also discuss how operating system specific environment variables may be accessed and modified from a GAMS program.

#### 4.40.2.1 Double Dash Parameters and Compile-Time Variables: A Simple Example

We will introduce double dash parameters and compile-time variables using as an example the well-known transportation model [TRNSPORT]. Assume we wish to explore how the solution changes if the demand assumes various values. To model this, we will introduce the compile-time variable DMULT, a multiplier for the demand b:

```gams
$set DMULT 1
...
demand(j) .. sum(i, x(i,j)) =g= %DMULT% * b(j) ;
```

Note that DMULT is defined with the dollar control option $set and its value is set to 1, which corresponds to the base line. Observe that the compile-time variable is referenced in the equation with the %...% notation. %DMULT% will be replaced at compile time by its value, in this case 1. We are now in the position to run the program multiple times by just changing the value of DMULT in the first line, which will automatically change the multiplier in the equation to the respective value.

GAMS offers a more convenient way for setting a compile-time variable like DMULT to a variety of values:

```gams
$if not set DMULT $set DMULT 1
...
demand(j) .. sum(i, x(i,j)) =g= %DMULT% * b(j) ;
```

Note that we set a default value for DMULT using conditional compilation. We may change this default on the command line when calling GAMS by specifying DMULT as a double dash parameter as follows:

```bash
> gams trnsport.gms --DMULT=0.9
```

Observe that with the specification --DMULT=0.9 the compile-time variable DMULT is set and therefore the default does not apply. Thus the double dash parameter facilitates changing the value of a compile-time variable directly on the command line as part of the GAMS call while the respective GAMS file remains unchanged.

Assume that the model contains a second compile-time variable called METHOD that acts as a switch for various methods of solving the model and it may take the values 1, 2 and 3. In this case both compile-time variables may be set on the command line as follows:

```bash
> gams trnsport.gms --DMULT=1.12 --METHOD=3
```

In the next two subsections we will discuss double dash parameters and compile-time variables in more detail.
Double dash parameters are user-defined command line parameters that are used to define scoped compile-time variables or to assign values to scoped compile-time variables. The general syntax is:

> gams myfile --NAME=value

Here \textit{NAME} is the name of the double dash parameter and \textit{value} is its assigned value that may be any string. If the string contains spaces or other token terminating characters, the string value should be quoted. Consider the following simple example:

> gams myfile --keycity=Boston --myvalue=7.6 --dothis="display x;"

Suppose that \textit{myfile.gms} contains the following lines with the scoped compile-time variables \textit{keycity}, \textit{myvalue} and \textit{dothis}:

\begin{verbatim}
x("%keycity%")=%myvalue%;
%dothis%
\end{verbatim}

Note that the compile-time variables are referenced using the notation \%...\%. The GAMS call above has the effect that at compile time the compile-time variables are substituted with the values specified through the double dash parameters resulting in the following:

\begin{verbatim}
x("Boston")=7.6;
display x;
\end{verbatim}

GAMS offers three alternative syntax variants for defining double dash parameters:

> gams myfile //NAME=value
> gams myfile /-NAME=value
> gams myfile -/NAME=value

Note that the four syntax variants may be used interchangeably and have the same effect.

Double dash parameters are particularly useful for specifying granularity when modeling a discretization of time and or space. For example, in the model \textit{[CHAIN]} the problem is to find the chain with minimal potential energy, assuming the chain has uniform density, is suspended between two points and has a given length \(L\). Consider the following code snippet:

\begin{verbatim}
$if not set nh $set nh 50
Set nh / i0*i%nh% /;
\end{verbatim}

The first two lines use \textit{conditional compilation} to set a default value for the compile-time variable \textit{nh}. Note that the value of \textit{nh} determines the cardinality of the set \textit{nh}, which is used for the discretization. The value of \textit{nh} may be easily set on the command line to any desired value using \textit{nh} as a double dash parameter:

> gams chain --nh=100
Note that the double dash functionality supersedes the command line parameters user1, ..., user5 which are accessible in the source file via %gams.user1%, ..., %gams.user5%. The example above would work with user1 in the following way:

\[
\begin{align*}
\$set\ nh\ 50 \\
\$if\ not\ "%gams.user1"==""\$set\ nh\ %gams.user1\% \\
Set\ nh\ /\ i0*i%nh%/ ;
\end{align*}
\]

The modified code could be called with:

> gams chain user1=100

Note that the double dash parameters facilitate using meaningful names instead of the generic names user1, ..., user5. This is especially useful if there are multiple parameters to pass on to the GAMS program.

Observe that the dollar control option $setDDList may be used to ensure that a model can only be run with the listed double dash parameters:

\[
\begin{align*}
\$setDDList\ nh \\
\$if\ not\ errorFree\$log ***\ Only\ allowed\ double\ dash\ options\ is:\ --nh=value
\end{align*}
\]

### 4.40.2.3 Compile-Time Variables

Compile-time variables are special variables that are substituted with their values at compile-time. They are not declared and defined with regular declaration statements like standard symbols (sets, parameters, ...), but they are defined with the dollar control option $set and its variants. There are three kinds of compile-time variables that differ in their scope level: local, scoped and global. An overview is given in Table 1.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Availability</th>
<th>Defined with</th>
<th>Removed from the system with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>Available only in the input file where they are defined</td>
<td>$setLocal.</td>
<td>$dropLocal</td>
</tr>
<tr>
<td>Scoped</td>
<td>Available in the input file where they are defined and in all include files of the input file.</td>
<td>$set</td>
<td>$drop</td>
</tr>
<tr>
<td>Global</td>
<td>Available in the input file where they are defined, in all parent files and in all include files.</td>
<td>$setGlobal</td>
<td>$dropGlobal</td>
</tr>
</tbody>
</table>

**Table 1**: Scope Levels for Compile-Time Variables in GAMS

Note that scoped compile-time variables may also be defined on the command line with double dash parameters. For example, in the example above the compile-time variable DMULT may be referenced in the equation demand without being defined with the dollar control option $set as long as it is defined and set on the command line. Note further, that global compile-time variables are saved in work files.

While the scope of a compile-time variable cannot be directly changed, but dropping and adding variables in different scopes accomplishes the same. Consider the following example:
Note that the compile-time variable MYVAR is first defined as a scoped variable with the value xxx. The dollar control option ifThen tests whether MYVAR was defined with $set and since this is TRUE the next four dollar control statements are processed: a new global compile-time variable called MYVAR is defined and is set to the value of the scoped compile-time variable MYVAR, the resulting compile-time variables are shown, the scoped compile-time variable is removed from the system with the option $drop and the resulting compile-time variables are shown again. A similar procedure is followed to change the global compile-time variable MYVAR to a local compile-time variable with the same name and value.

The output generated by the four dollar control options $show follows:

<table>
<thead>
<tr>
<th>Level</th>
<th>SetVal</th>
<th>Type</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>SCOPED</td>
<td>xxx</td>
</tr>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>GLOBAL</td>
<td>xxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>SetVal</th>
<th>Type</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>GLOBAL</td>
<td>xxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>SetVal</th>
<th>Type</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>LOCAL</td>
<td>xxx</td>
</tr>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>GLOBAL</td>
<td>xxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>SetVal</th>
<th>Type</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MYVAR</td>
<td>LOCAL</td>
<td>xxx</td>
</tr>
</tbody>
</table>

Observe that this report is called Environment Report. This name is unfortunate, since the variables reported are in fact compile-time variables. Environment variables in GAMS are discussed in section Environment Variables in GAMS below.

Note that if a compile-time variable is referenced with %MYVAR%, it could reference a global, scoped or local compile-time variable called MYVAR. GAMS will always access the compile-time variable MYVAR with the most local scope. Thus if all three scopes are defined, the local compile-time variable is accessed first, then the scoped and then the global, as demonstrated in the following example:
The resulting log output will be:

```
local
scoped
global
%XXX%
```

Note that how `%XXX%` will be handled if no compile-time variable `XXX` is defined, is determined by the command line parameter `stringChk`.

For a full list of dollar control options that affect compile-time variables, see section Dollar Control Options for Compile-Time Variables and Environment Variables.

### 4.40.2.4 Environment Variables in GAMS

GAMS programs have access to operating systems via `%sysEnv.NAME%`. Operating system environment variables may be modified or new environment variables may be defined with the dollar control option `$setEnv`. Consider the following artificial example:

```
$log %sysEnv.GEORGE%
$setEnv GEORGE Dantzig
$log %sysEnv.GEORGE%
$dropEnv GEORGE
$log %sysEnv.GEORGE%
```

Note that the dollar control option `$dropEnv` removes an environment variable. The log output follows:

```
%sysEnv.GEORGE%
Dantzig
%sysEnv.GEORGE%
```

There are two environment variables in GAMS that are specific to the GDX facility: `GDXCONVERT` and `GDXCOMPRESS`. Since GDX is used by utilities and other programs some general customization can be achieved via these environment variables. Their values determine the type of GDX files that are written. These environment variables may be overwritten through the command line parameters `gdxConvert` and `gdxCompress` or inside the GAMS file with the dollar control option `$setEnv`. Consider the following example that uses the latter functionality:
Scalar x /1/;
$log %sysEnv.GDXCONVERT%
$log %sysEnv.GDXCOMPRESS%
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"

$setEnv GDXCONVERT v6
$setEnv GDXCOMPRESS 1
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"

$setEnv GDXCONVERT v7
$setEnv GDXCOMPRESS 0
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"

Note that gdxdump is a **GDX utility** that writes the contents of a GDX file as a GAMS formatted text file. The switch -v lets gdxdump print the file version information. With the grep utility we filter the lines that contain either File format or Compression. This code is run with the following call that initializes GDXCOMPRESS and GDXCONVERT:

> gams gdxenv.gms gdxCompress=0 gdxConvert=v6 lo=3

The output follows:

--- Starting compilation
v6
0
--- gdxenv.gms(7) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format : 6
* Compression : 0
--- gdxenv.gms(14) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format : 6
* Compression : 1
--- gdxenv.gms(21) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format : 7
* Compression : 0
--- gdxenv.gms(22) 2 Mb
--- Starting execution - empty program

### 4.40.3 Compile-Time Constants

Compile-time constants are constants that are related to some **functions, model attributes or options**. They have a fixed value and are referenced as %prefix.constant%. Here prefix is the name of the respective function, model attribute or option and constant is the name of the constant.

For example, the function handleStatus is used in the context of grid computing. Typically, a **collection loop** may take the following form:
loop(pp$(handleStatus(h(pp)) = 2), ... );

Alternatively, the following formulation may be used:

\[
\text{loop}(pp$(\text{handleStatus}(h(pp)=%\text{handleStatus.ready}%), ... );
\]

Observe that the compile-time constant \%\text{handleStatus.ready}\% equals the value of 2. See the table below for other compile-time constants that are related to the function \text{handleStatus}.

Note

Compile-time constants are replaced at compile time and cannot be manipulated or reassigned.

Though compile-time constants are most often used in the context of the function, model attribute or option indicated with the prefix, they are in fact context free and may be used anywhere where an integer is expected. Consider the following example:

Scalar \( x / %\text{solPrint.on}% /; \) display \( x; \)

A complete list of the compile-time constants is given in Table 2.

\textit{Table 2: Compile-Time Constants}

<table>
<thead>
<tr>
<th>Compile-Time Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%\text{handleStatus.Unknown}%</td>
<td>0</td>
</tr>
<tr>
<td>%\text{handleStatus.running}%</td>
<td>1</td>
</tr>
<tr>
<td>%\text{handleStatus.ready}%</td>
<td>2</td>
</tr>
<tr>
<td>%\text{handleStatus.failure}%</td>
<td>3</td>
</tr>
<tr>
<td>%\text{modelStat.optimal}%</td>
<td>1</td>
</tr>
<tr>
<td>%\text{modelStat.locally Optimal}%</td>
<td>2</td>
</tr>
<tr>
<td>%\text{modelStat.unbounded}%</td>
<td>3</td>
</tr>
<tr>
<td>%\text{modelStat.infeasible}%</td>
<td>4</td>
</tr>
<tr>
<td>%\text{modelStat.locallyInfeasible}%</td>
<td>5</td>
</tr>
<tr>
<td>%\text{modelStat.intermediateInfeasible}%</td>
<td>6</td>
</tr>
<tr>
<td>%\text{modelStat.feasibleSolution}%</td>
<td>7</td>
</tr>
<tr>
<td>%\text{modelStat.integerSolution}%</td>
<td>8</td>
</tr>
<tr>
<td>%\text{modelStat.intermediateNon-Integer}%</td>
<td>9</td>
</tr>
<tr>
<td>%\text{modelStat.integerInfeasible}%</td>
<td>10</td>
</tr>
<tr>
<td>%\text{modelStat.licensingProblem}%</td>
<td>11</td>
</tr>
<tr>
<td>%\text{modelStat.errorUnknown}%</td>
<td>12</td>
</tr>
<tr>
<td>%\text{modelStat.errorNoSolution}%</td>
<td>13</td>
</tr>
<tr>
<td>%\text{modelStat.noSolutionReturned}%</td>
<td>14</td>
</tr>
<tr>
<td>%\text{modelStat.solvedUnique}%</td>
<td>15</td>
</tr>
<tr>
<td>%\text{modelStat.solved}%</td>
<td>16</td>
</tr>
<tr>
<td>%\text{modelStat.solvedSingular}%</td>
<td>17</td>
</tr>
<tr>
<td>%\text{modelStat.unbounded-NoSolution}%</td>
<td>18</td>
</tr>
<tr>
<td>%\text{modelStat.infeasible-NoSolution}%</td>
<td>19</td>
</tr>
<tr>
<td>%\text{platformCode.Unknown}%</td>
<td>0</td>
</tr>
</tbody>
</table>
### Compile-Time Constant

<table>
<thead>
<tr>
<th>Compile-Time Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%platformCode.DEG%</td>
<td>1</td>
</tr>
<tr>
<td>%platformCode.LEG%</td>
<td>2</td>
</tr>
<tr>
<td>%platformCode.WEX%</td>
<td>3</td>
</tr>
<tr>
<td>%solPrint.off%</td>
<td>0</td>
</tr>
<tr>
<td>%solPrint.on%</td>
<td>1</td>
</tr>
<tr>
<td>%solPrint.silent%</td>
<td>2</td>
</tr>
<tr>
<td>%solPrint.summary% (deprecated)</td>
<td>0</td>
</tr>
<tr>
<td>%solPrint.report% (deprecated)</td>
<td>1</td>
</tr>
<tr>
<td>%solPrint.quiet% (deprecated)</td>
<td>2</td>
</tr>
<tr>
<td>%solveLink.chainScript%</td>
<td>0</td>
</tr>
<tr>
<td>%solveLink.callScript%</td>
<td>1</td>
</tr>
<tr>
<td>%solveLink.callModule%</td>
<td>2</td>
</tr>
<tr>
<td>%solveLink.aSyncGrid%</td>
<td>3</td>
</tr>
<tr>
<td>%solveLink.aSyncSimulate%</td>
<td>4</td>
</tr>
<tr>
<td>%solveLink.loadLibrary%</td>
<td>5</td>
</tr>
<tr>
<td>%solveLink.aSyncThreads%</td>
<td>6</td>
</tr>
<tr>
<td>%solveLink.threadsSimulate%</td>
<td>7</td>
</tr>
<tr>
<td>%solveOpt.replace%</td>
<td>0</td>
</tr>
<tr>
<td>%solveOpt.merge%</td>
<td>1</td>
</tr>
<tr>
<td>%solveOpt.clear%</td>
<td>2</td>
</tr>
<tr>
<td>%solveStat.normalCompletion%</td>
<td>1</td>
</tr>
<tr>
<td>%solveStat.iterationInterrupt%</td>
<td>2</td>
</tr>
<tr>
<td>%solveStat.resourceInterrupt%</td>
<td>3</td>
</tr>
<tr>
<td>%solveStat.terminatedBySolver%</td>
<td>4</td>
</tr>
<tr>
<td>%solveStat.evaluationInterrupt%</td>
<td>5</td>
</tr>
<tr>
<td>%solveStat.capabilityProblems%</td>
<td>6</td>
</tr>
<tr>
<td>%solveStat.licensingProblems%</td>
<td>7</td>
</tr>
<tr>
<td>%solveStat.userInterrupt%</td>
<td>8</td>
</tr>
<tr>
<td>%solveStat.setupFailure%</td>
<td>9</td>
</tr>
<tr>
<td>%solveStat.solverFailure%</td>
<td>10</td>
</tr>
<tr>
<td>%solveStat.internalSolverFailure%</td>
<td>11</td>
</tr>
<tr>
<td>%solveStat.solveProcessingSkipped%</td>
<td>12</td>
</tr>
<tr>
<td>%solveStat.systemFailure%</td>
<td>13</td>
</tr>
</tbody>
</table>

### 4.40.4 GAMS Compile Time and Execution Time Phase

The GAMS log indicates different phases of a job run in the log file:

```plaintext
...  
--- Starting compilation  
...  
--- Starting execution: elapsed 0:00:00.056  
...  
```

During compilation GAMS converts the GAMS user program into lower-level instructions that are executed during execution time. Before the user program is converted into lower-level instructions the compiler
processes the input: the compile time variables and macros are substituted and comments are removed. Moreover, during compilation any dollar control option present in the user code is executed. Many of these dollar control options impact the behavior of the compilation phase (e.g. $include instructs the compiler to process a file before continuing processing the remaining part of the current file). The code the compiler actually converts into lower-level instructions is echoed (by default) to the listing file. The compiler also assembles the list of user symbols (sets, parameters, variables, ...) and the list of labels. These lists become immutable after the compiler finishes. So during execution time, for example, no new labels can be added. The only exception from this is during a continued compilation/execution using the save and restart facility. The separation between compile time and execution time is confusing especially for novice users and mistakes as the following are frequent:

```gams
file fInput / data.txt /;
scalar iCnt; for (iCnt=1 to 100, put fInput iCnt:0:0 /); putClose fInput;
set i /
$include data.txt
/;
```

The intention of the code, that does not work, is clear: The put facility is used to create the input file data.txt that is included via the $include instruction. The problem with this code is that the $include instruction is executed at compile time while the code using the put statement is executed at execution time, i.e. after the compilation phase is over. Hence, the compiler tries to include this file before the put instructions are executed. If the file data.txt is not present, the compiler will terminate with a compilation error, but if a file with name data.txt is present this one will be processed by the $include and a mistake like this might remain undiscovered for a long time.

Most of the time GAMS performs compilation and execution in one GAMS job which makes it even harder to grasp the concept of compile and execution time. The command line parameter action can be used to separate the compilation and execution phase into multiple jobs.

### 4.40.5 List of Command Line Parameters

In the following two subsections we will present an overview of the command line parameters with brief descriptions. Detailed descriptions of all command line parameters follow in section Detailed Descriptions of All Options below.

#### 4.40.5.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>GAMS processing request</td>
</tr>
<tr>
<td>appendExpand</td>
<td>Expand file append option</td>
</tr>
<tr>
<td>appendLog</td>
<td>Log file append option</td>
</tr>
<tr>
<td>appendOut</td>
<td>Output file append option</td>
</tr>
<tr>
<td>asyncSolLst</td>
<td>Print solution listing when asynchronous solve (Grid or Threads) is used</td>
</tr>
<tr>
<td>captureModelInstance</td>
<td>Switch to capture all model instances within a run</td>
</tr>
<tr>
<td>case</td>
<td>Output case option for LST file</td>
</tr>
<tr>
<td>cErr</td>
<td>Compile time error limit</td>
</tr>
<tr>
<td>charSet</td>
<td>Character set flag</td>
</tr>
<tr>
<td>checkErrorLevel</td>
<td>Check errorLevel automatically after executing external program</td>
</tr>
<tr>
<td>connectIn</td>
<td>Specify YAML Connect script file processed at start of GAMS</td>
</tr>
<tr>
<td>connectOut</td>
<td>Specify YAML Connect script file processed at end of GAMS</td>
</tr>
<tr>
<td>curDir</td>
<td>Current directory</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>decryptKey</td>
<td>Key to decrypt a text file that was encrypted via <code>$encrypt</code></td>
</tr>
<tr>
<td>dFormat</td>
<td>Date format</td>
</tr>
<tr>
<td>digit</td>
<td>Switch default for &quot;$on/offDigit&quot;</td>
</tr>
<tr>
<td>docFile</td>
<td>Filename stem for documentation files</td>
</tr>
<tr>
<td>dumpOpt</td>
<td>Writes preprocessed input to the file input.dmp</td>
</tr>
<tr>
<td>dumpOptGDX</td>
<td>Defines a GDX file name stem created when using DumpOpt</td>
</tr>
<tr>
<td>dumpParms</td>
<td>GAMS parameter logging</td>
</tr>
<tr>
<td>dumpParmsLogPrefix</td>
<td>Prefix of lines triggered by DumpParms&gt;1</td>
</tr>
<tr>
<td>ECImplicitLoad</td>
<td>Allow implicit loading of symbols from embedded code or not</td>
</tr>
<tr>
<td>empty</td>
<td>Switch default for &quot;$on/offEmpty&quot;</td>
</tr>
<tr>
<td>encryptKey</td>
<td>Key to encrypt a text file using <code>$encrypt</code></td>
</tr>
<tr>
<td>eolCom</td>
<td>Switch default for &quot;$on/offEolCom&quot; and &quot;$eolCom&quot;</td>
</tr>
<tr>
<td>eolOnly</td>
<td>Single key-value pairs (immediate switch)</td>
</tr>
<tr>
<td>epsToZero</td>
<td>Treat eps as zero when unloading to GDX</td>
</tr>
<tr>
<td>errMsg</td>
<td>Placing of compilation error messages</td>
</tr>
<tr>
<td>errNam</td>
<td>Name of error message file</td>
</tr>
<tr>
<td>error</td>
<td>Force a compilation error with message</td>
</tr>
<tr>
<td>errorLog</td>
<td>Max error message lines written to the log for each error</td>
</tr>
<tr>
<td>ctLim</td>
<td>Elapsed time limit in seconds</td>
</tr>
<tr>
<td>execMode</td>
<td>Limits on external programs that are allowed to be executed</td>
</tr>
<tr>
<td>expand</td>
<td>Expanded (include) input file name</td>
</tr>
<tr>
<td>fdDelta</td>
<td>Step size for finite differences</td>
</tr>
<tr>
<td>fdOpt</td>
<td>Options for finite differences</td>
</tr>
<tr>
<td>fErr</td>
<td>Alternative error message file</td>
</tr>
<tr>
<td>fileCase</td>
<td>Casing of file names and paths (put, gdx, ref, $include, etc.)</td>
</tr>
<tr>
<td>fileStem</td>
<td>Sets the file stem for output files which use the input file name as stem by default</td>
</tr>
<tr>
<td>fileStemApFromEnv</td>
<td>Append a string read from an environment variable to the &quot;FileStem&quot;</td>
</tr>
<tr>
<td>filtered</td>
<td>Switch default for &quot;$on/offFiltered&quot;</td>
</tr>
<tr>
<td>forceWork</td>
<td>Force GAMS to process a save file created with a newer GAMS version or with execution errors</td>
</tr>
<tr>
<td>forLim</td>
<td>GAMS looping limit</td>
</tr>
<tr>
<td>G205</td>
<td>Use GAMS version 2.05 syntax</td>
</tr>
<tr>
<td>GDX</td>
<td>GAMS data exchange file name</td>
</tr>
<tr>
<td>gdxCompress</td>
<td>Compression of generated GDX file</td>
</tr>
<tr>
<td>gdxConvert</td>
<td>Version of GDX files generated (for backward compatibility)</td>
</tr>
<tr>
<td>gdxUels</td>
<td>Unload labels or UELs to GDX either squeezed or full</td>
</tr>
<tr>
<td>gridDir</td>
<td>Grid file directory</td>
</tr>
<tr>
<td>gridScript</td>
<td>Grid submission script</td>
</tr>
<tr>
<td>heapLimit</td>
<td>Maximum Heap size allowed in MB</td>
</tr>
<tr>
<td>IDCGenerateGDXInput</td>
<td>GDX file name with data for implicit input</td>
</tr>
<tr>
<td>IDCGenerateGDXOutput</td>
<td>GDX file name for data for implicit output</td>
</tr>
<tr>
<td>IDCGenerateGDX</td>
<td>Specify GDX file name of input and output side of data contract</td>
</tr>
<tr>
<td>IDCGenerateGDXInput</td>
<td>Specify GDX file name of input side of data contract</td>
</tr>
<tr>
<td>IDCGenerateGDXOutput</td>
<td>Specify GDX file name of output side of data contract</td>
</tr>
<tr>
<td>IDCGenerateJSON</td>
<td>Specify JSON file name of data contract</td>
</tr>
<tr>
<td>IDCJSON</td>
<td>Specify JSON file name to verify data contract</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IDCProtect</td>
<td>Flag to control assignment protection of external input symbols</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment flag</td>
</tr>
<tr>
<td>inlineCom</td>
<td>Switch default for &quot;$on/offInline&quot; and &quot;$inlineCom&quot;</td>
</tr>
<tr>
<td>input</td>
<td>Input file</td>
</tr>
<tr>
<td>inputDir, inputDir1..40</td>
<td>Input file directories</td>
</tr>
<tr>
<td>interactiveSolver</td>
<td>Allow solver to interact via command line input</td>
</tr>
<tr>
<td>jobTrace</td>
<td>Job trace string to be written to the trace file at the end of a GAMS job</td>
</tr>
<tr>
<td>keep</td>
<td>Controls keeping or deletion of process directory and scratch files</td>
</tr>
<tr>
<td>libIncDir</td>
<td>LibInclude directory</td>
</tr>
<tr>
<td>license</td>
<td>Use alternative license file</td>
</tr>
<tr>
<td>listing</td>
<td>Switch default for &quot;$on/offListing&quot;</td>
</tr>
<tr>
<td>logFile</td>
<td>Log file name</td>
</tr>
<tr>
<td>logLine</td>
<td>Amount of line tracing to the log file</td>
</tr>
<tr>
<td>logOption</td>
<td>Log option</td>
</tr>
<tr>
<td>lstTitleLeftAligned</td>
<td>Write title of LST file all left aligned</td>
</tr>
<tr>
<td>maxExecError</td>
<td>Execution time error limit</td>
</tr>
<tr>
<td>maxGenericFiles</td>
<td>Maximum number of generic file names tried at execution time file creation</td>
</tr>
<tr>
<td>maxProcDir</td>
<td>Maximum number of 225+ process directories</td>
</tr>
<tr>
<td>MCPRHoldFx</td>
<td>Print list of rows that are perpendicular to variables removed due to the holdfixed setting</td>
</tr>
<tr>
<td>multi</td>
<td>Switch default for &quot;$on/offMulti[R]&quot;</td>
</tr>
<tr>
<td>multiPass</td>
<td>Multipass facility</td>
</tr>
<tr>
<td>noNewVarEqu</td>
<td>Triggers a compilation error when new equations or variable symbols are introduced</td>
</tr>
<tr>
<td>on115</td>
<td>Generate errors for unknown unique element in an equation</td>
</tr>
<tr>
<td>output</td>
<td>Listing file name</td>
</tr>
<tr>
<td>pageContr</td>
<td>Output file page control option</td>
</tr>
<tr>
<td>pageSize</td>
<td>Output file page size (=0 no paging)</td>
</tr>
<tr>
<td>pageWidth</td>
<td>Output file page width</td>
</tr>
<tr>
<td>parmFile</td>
<td>Command Line Parameter include file</td>
</tr>
<tr>
<td>pLicense</td>
<td>Privacy license file name</td>
</tr>
<tr>
<td>prefixLoadPath</td>
<td>Prepend GAMS system directory to library load path</td>
</tr>
<tr>
<td>previousWork</td>
<td>Indicator for writing workfile with previous workfile version</td>
</tr>
<tr>
<td>procDir</td>
<td>Process Directory</td>
</tr>
<tr>
<td>procDirPath</td>
<td>Directory to create process directory in</td>
</tr>
<tr>
<td>procTreeMemMonitor</td>
<td>Monitor the memory used by the GAMS process tree</td>
</tr>
<tr>
<td>procTreeMemTicks</td>
<td>Set wait interval between memory monitor checks: ticks = milliseconds</td>
</tr>
<tr>
<td>profile</td>
<td>Execution profiling</td>
</tr>
<tr>
<td>profileFile</td>
<td>Write profile information to this file</td>
</tr>
<tr>
<td>profileTol</td>
<td>Minimum time a statement must use to appear in profile generated output</td>
</tr>
<tr>
<td>putDir</td>
<td>Put file directory</td>
</tr>
<tr>
<td>putND</td>
<td>Number of decimals for put files</td>
</tr>
<tr>
<td>putNR</td>
<td>Numeric round format for put files</td>
</tr>
<tr>
<td>putPS</td>
<td>Page size for put files</td>
</tr>
<tr>
<td>putPW</td>
<td>Page width for put files</td>
</tr>
<tr>
<td>pyMultInst</td>
<td>GAMS/Python Multiple Instance Interpreter</td>
</tr>
<tr>
<td>reference</td>
<td>Symbol reference file</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>referenceLineNo</td>
<td>Controls the line numbers written to a reference file</td>
</tr>
<tr>
<td>scrDir</td>
<td>Scratch directory</td>
</tr>
<tr>
<td>scrExt</td>
<td>Scratch file extension to be used with temporary files</td>
</tr>
<tr>
<td>scrNam</td>
<td>Work file names stem</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed</td>
</tr>
<tr>
<td>showOSMemory</td>
<td>Show the memory usage reported by the Operating System instead of the</td>
</tr>
<tr>
<td></td>
<td>internal counting</td>
</tr>
<tr>
<td>stepSum</td>
<td>Summary of computing resources used by job steps</td>
</tr>
<tr>
<td>strictSingleton</td>
<td>Error if assignment to singleton set has multiple elements</td>
</tr>
<tr>
<td>stringChk</td>
<td>String substitution options</td>
</tr>
<tr>
<td>suffixDLVars</td>
<td>Switch default for &quot;$on/offSuffixDLVars&quot;</td>
</tr>
<tr>
<td>suppress</td>
<td>Compiler listing option</td>
</tr>
<tr>
<td>symbol</td>
<td>Symbol table file</td>
</tr>
<tr>
<td>sys10</td>
<td>Changes rpower to ipower when the exponent is constant and within 1e-12</td>
</tr>
<tr>
<td></td>
<td>of an integer</td>
</tr>
<tr>
<td>sys11</td>
<td>Dynamic resorting if indices in assignment/data statements are not in</td>
</tr>
<tr>
<td></td>
<td>natural order</td>
</tr>
<tr>
<td>sys12</td>
<td>Pass model with generation errors to solver</td>
</tr>
<tr>
<td>sys15</td>
<td>Automatic switching of data structures used in search records</td>
</tr>
<tr>
<td>sys16</td>
<td>Disable search record memory (aka execute this as pre-GAMS 24.5)</td>
</tr>
<tr>
<td>sys17</td>
<td>Disable sparsity trees growing with permutation (aka execute this as</td>
</tr>
<tr>
<td></td>
<td>pre-GAMS 24.5)</td>
</tr>
<tr>
<td>sys18</td>
<td>Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-</td>
</tr>
<tr>
<td></td>
<td>point numbers</td>
</tr>
<tr>
<td>sys19</td>
<td>Disable permutation on Column Generation (aka execute this as pre-GAMS</td>
</tr>
<tr>
<td></td>
<td>36)</td>
</tr>
<tr>
<td>sysDir</td>
<td>GAMS system directory where GAMS executables reside</td>
</tr>
<tr>
<td>sysIncDir</td>
<td>SysInclude directory</td>
</tr>
<tr>
<td>tabIn</td>
<td>Tab spacing</td>
</tr>
<tr>
<td>tFormat</td>
<td>Time format</td>
</tr>
<tr>
<td>threadsAsync</td>
<td>Limit on number of threads to be used for asynchronous solves (solveLink=6)</td>
</tr>
<tr>
<td>timer</td>
<td>Instruction timer threshold in milli seconds</td>
</tr>
<tr>
<td>trace</td>
<td>Trace file name</td>
</tr>
<tr>
<td>traceLevel</td>
<td>Modelstat/Solvestat threshold used in conjunction with action=GT</td>
</tr>
<tr>
<td>traceOpt</td>
<td>Trace file format option</td>
</tr>
<tr>
<td>user1..5</td>
<td>User strings</td>
</tr>
<tr>
<td>warnings</td>
<td>Number of warnings permitted before a run terminates</td>
</tr>
<tr>
<td>workDir</td>
<td>Working directory</td>
</tr>
<tr>
<td>zeroRes</td>
<td>The results of certain operations will be set to zero if abs(result) LE</td>
</tr>
<tr>
<td></td>
<td>ZeroRes</td>
</tr>
<tr>
<td>zeroResRep</td>
<td>Report underflow as a warning when abs(results) LE ZeroRes and result set</td>
</tr>
<tr>
<td></td>
<td>to zero</td>
</tr>
</tbody>
</table>

### 4.40.5.2 Solver-Related Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRatio</td>
<td>Basis detection threshold</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CNS</td>
<td>Constrained Nonlinear Systems - default solver</td>
</tr>
<tr>
<td>DNLP</td>
<td>Non-Linear Programming with Discontinuous Derivatives - default solver</td>
</tr>
<tr>
<td>domLim</td>
<td>Domain violation limit solver default</td>
</tr>
<tr>
<td>EMP</td>
<td>Extended Mathematical Programs - default solver</td>
</tr>
<tr>
<td>forceOptFile</td>
<td>Overwrites other option file section mechanism</td>
</tr>
<tr>
<td>holdFixed</td>
<td>Treat fixed variables as constants</td>
</tr>
<tr>
<td>holdFixedAsync</td>
<td>Allow HoldFixed for models solved asynchronously as well</td>
</tr>
<tr>
<td>integer1..5</td>
<td>Integer communication cells</td>
</tr>
<tr>
<td>intVarUp</td>
<td>Set mode for default upper bounds on integer variables</td>
</tr>
<tr>
<td>iterLim</td>
<td>Iteration limit of solver</td>
</tr>
<tr>
<td>limCol</td>
<td>Maximum number of columns listed in one variable block</td>
</tr>
<tr>
<td>limRow</td>
<td>Maximum number of rows listed in one equation block</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming - default solver</td>
</tr>
<tr>
<td>MCP</td>
<td>Mixed Complementarity Problems - default solver</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed-Integer Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed-Integer Programming - default solver</td>
</tr>
<tr>
<td>MIQCP</td>
<td>Mixed Integer Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>NLP</td>
<td>Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>nodLim</td>
<td>Node limit in branch and bound tree</td>
</tr>
<tr>
<td>optCA</td>
<td>Absolute Optimality criterion solver default</td>
</tr>
<tr>
<td>optCR</td>
<td>Relative Optimality criterion solver default</td>
</tr>
<tr>
<td>optDir</td>
<td>Option file directory</td>
</tr>
<tr>
<td>optFile</td>
<td>Default option file</td>
</tr>
<tr>
<td>QCP</td>
<td>Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>resLim</td>
<td>Wall-clock time limit for solver</td>
</tr>
<tr>
<td>RMINLP</td>
<td>Relaxed Mixed-Integer Non-Linear Programming - default solver</td>
</tr>
<tr>
<td>RMIP</td>
<td>Relaxed Mixed-Integer Programming - default solver</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>Relaxed Mixed Integer Quadratically Constrained Programs - default solver</td>
</tr>
<tr>
<td>RMPEC</td>
<td>Relaxed Mathematical Programs with Equilibrium Constraints - default solver</td>
</tr>
<tr>
<td>savePoint</td>
<td>Save solver point in GDX file</td>
</tr>
<tr>
<td>scriptExit</td>
<td>Program or script to be executed at the end of a GAMS run</td>
</tr>
<tr>
<td>scriptFirst</td>
<td>First line to be written to GAMSNEXT file</td>
</tr>
<tr>
<td>solPrint</td>
<td>Solution report print option</td>
</tr>
<tr>
<td>solveLink</td>
<td>Solver link option</td>
</tr>
<tr>
<td>solver</td>
<td>Default solver for all model types that the solver is capable to process</td>
</tr>
<tr>
<td>solverCntr</td>
<td>Solver control file name</td>
</tr>
<tr>
<td>solverDict</td>
<td>Solver dictionary file name</td>
</tr>
<tr>
<td>solverInst</td>
<td>Solver instruction file name</td>
</tr>
<tr>
<td>solverMatr</td>
<td>Solver matrix file name</td>
</tr>
<tr>
<td>solverSolu</td>
<td>Solver solution file name</td>
</tr>
<tr>
<td>solverStat</td>
<td>Solver status file name</td>
</tr>
<tr>
<td>subSys</td>
<td>Name of subsystem configuration file</td>
</tr>
<tr>
<td>sysOut</td>
<td>Solver Status file reporting option</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads to be used by a solver</td>
</tr>
<tr>
<td>workFactor</td>
<td>Memory Estimate multiplier for some solvers</td>
</tr>
<tr>
<td>workSpace</td>
<td>Work space for some solvers in MB</td>
</tr>
</tbody>
</table>
### 4.40.5.3 Save and Restart Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSave</td>
<td>Creates a forced work file, i.e., the file is saved even if execution errors or other errors occurred</td>
</tr>
<tr>
<td>restart</td>
<td>Name of a restart file, see The Save and Restart Feature</td>
</tr>
<tr>
<td>restartNamed</td>
<td>Name of another matching restart file, see Obfuscated Work Files</td>
</tr>
<tr>
<td>save</td>
<td>Creates a work file, see The Save and Restart Feature</td>
</tr>
<tr>
<td>saveObfuscate</td>
<td>Creates an obfuscated work file, see Obfuscated Work Files</td>
</tr>
<tr>
<td>symPrefix</td>
<td>Prefix all symbols encountered during compilation by the specified string in work file</td>
</tr>
<tr>
<td>xSave</td>
<td>Creates a compressed work file</td>
</tr>
<tr>
<td>xSaveObfuscate</td>
<td>Creates a compressed obfuscated work file</td>
</tr>
</tbody>
</table>

### 4.40.6 Detailed Descriptions of All Options

In this section we will give detailed descriptions of all options that may be used as command line parameters, in option statements or as model attributes. Note that we indicate for each entry the context in which the option is available. Observe that the options are listed in alphabetical order for easy reference.

**action (string):** GAMS processing request

- **Synonym:** A
- **Available:** Command line

This option controls the way GAMS processes the input file. In particular GAMS currently processes the input file in multiple phases and this allows one to restrict the phases used. The two phases in order are:

- **Compilation** During this pass, the file is compiled, and syntax errors are checked for. Data initialization statements like scalar, parameter, and table statements are also processed during this stage.
- **Execution** During this stage, all execution time statements including assignments, loops, and solves are executed.

The special action GT is related to the creation of trace reports. See also option traceLevel for details.

- **Default:** CE

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Restart After Solve</td>
</tr>
<tr>
<td>C</td>
<td>CompileOnly</td>
</tr>
<tr>
<td>E</td>
<td>ExecuteOnly</td>
</tr>
<tr>
<td>CE</td>
<td>Compile and Execute</td>
</tr>
<tr>
<td>GT</td>
<td>Trace Report</td>
</tr>
</tbody>
</table>

**appendExpand (boolean):** Expand file append option

- **Synonym:** AE
Available: Command line

This option controls the manner of file opening of the option expand.

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset expand file</td>
</tr>
<tr>
<td>1</td>
<td>Append to expand file</td>
</tr>
</tbody>
</table>

**appendLog (boolean):** Log file append option

Synonym: AL

Available: Command line

This option is used in conjunction with the setting of logOption to 2 and 4, where the log from the GAMS run is redirected to a file. Setting this option to 1 will ensure that the log file is appended to and not overwritten (replaced).

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset log file</td>
</tr>
<tr>
<td>1</td>
<td>Append to logfile</td>
</tr>
</tbody>
</table>

**appendOut (boolean):** Output file append option

Synonym: AO

Available: Command line

Setting this option to 1 will ensure that the listing file is appended to and not overwritten (replaced).

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset listing file</td>
</tr>
<tr>
<td>1</td>
<td>Append to listing file</td>
</tr>
</tbody>
</table>

**asyncSolLst (boolean):** Print solution listing when asynchronous solve (Grid or Threads) is used

Available: Command line, Option statement

This option determines whether the solution listing is printed in the listing file when an asynchronous (grid or threads) solve is used and the function handleCollect or command execute_loadHandle successfully collect the results.

Default: 0
### The GAMS Call and Command Line Parameters

#### 4.40 Value | Meaning
--- | ---
0 | Do not print solution listing into lst file for asynchronous solves
1 | Print solution listing into lst file for asynchronous solves

**bRatio (real):** Basis detection threshold

Available: Command line, Option statement, Attribute statement (use before solve)

The `bRatio` value is used to detect whether the initial point (levels and marginals) passed to the solver represents a basis suitable for use by the solver.

Certain (pivotal) solution procedures can restart from an advanced basis that is constructed automatically using existing basis information, i.e. they do a *warm start*. This option is used to influence whether a warm start is done or not. GAMS provides a hint to the solver to suggest whether a basis can or might be extracted from the initial point. The hint is based on the number of rows with nonzero marginals and is computed internally as:

\[
\text{hint} := (0 == \text{bRatio}) \text{ or } (\text{rowsWithNonzeroMarg} > \text{nRows} \times \text{bratio})
\]

Note that setting `bRatio` to 1 causes the hint to be false, while setting `bRatio` to 0 causes the hint to be true. Note also that this is only a hint to the solver: some solvers (e.g. barrier methods) do not use this hint value.

Default: 0.25

**captureModelInstance (boolean):** Switch to capture all model instances within a run

Available: Command line

This option is a debugging option that helps to capture, if set to 1, the model instances that are generated and solved during the execution of the `solve` commands in a GAMS run. This is particularly important if one needs to make a model instance available to GAMS technical support to investigate a solver failure or analyze poor solver performance. The model instances are stored in separate files with names `gamsNNN.gms` where `NNN` is an increasing number. So sorting the files by time or name will correspond to the order in which these instances have been generated and solved. The format of the model instance is the GAMS scalar format and the instance is captured before the instance is solved by the selected solver, hence it contains the starting point but not the solution point.

Another way of capturing model instances becomes useful (and is triggered independent of this option) in case the user model does modularization by calling other GAMS programs that execute a solve statement via the GAMS executable. Since options are not automatically passed on to the GAMS subprocesses one might miss capturing some model instances. Moreover, due to constant folding and rounding in the scalar model format, the run of a scalar model instance might not reproduce a particular problematic issue of the original solve. Hence GAMS can capture the binary scratch files produced by the solve statement when executed under `solveLink=0` in a ZIP file. This is activated by setting a system environment variable named `ZIPSCRDIR_PGAMS` (e.g. `set ZIPSCRDIR_PGAMS=mymodel`). The content of this environment variable is used as a prefix for the ZIP file names, e.g. `mymodelNNN.zip` where `NNN` is an increasing number. Such ZIP files can help GAMS support staff to execute the model and have an increased chance to reproduce a problematic issue.

Note

- `CaptureModelInstance` cannot be used with `solveLink=6/7`. If this was set, GAMS will reset `solveLink` to 3/4 automatically.

Default: 0
### Value | Meaning
--- | ---
0 | Do not capture model instances
1 | Capture model instances

**case** (*boolean*): Output case option for LST file

Available: Command line

This option controls the case of the text in the listing file.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Write listing file in mixed case</td>
</tr>
<tr>
<td>1</td>
<td>Write listing file in upper case only</td>
</tr>
</tbody>
</table>

### cErr (*integer*): Compile time error limit

Available: Command line

The compilation will be aborted after n errors have occurred. By default, there is no error limit and GAMS compiles the entire input file and collects all the compilation errors that occur. If the file is too long and the compilation process is time consuming, **cerr** could be used to set to a low value while debugging the input file.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error limit (default)</td>
</tr>
<tr>
<td>n</td>
<td>Stop after n errors</td>
</tr>
</tbody>
</table>

### charSet (*boolean*): Character set flag

Available: Command line

This option specifies whether foreign language characters are permitted in comments and text items. For a list of standard GAMS characters, see table Legal Characters.

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use limited GAMS characters set</td>
</tr>
<tr>
<td>1</td>
<td>Accept any character in comments and text items (foreign language characters)</td>
</tr>
</tbody>
</table>

### cheat (*real*): Cheat value, i.e. minimum solution improvement threshold

Available: Attribute statement (use before solve)

For a branch-and-bound based solver, each new feasible solution must be at least the value of **cheat** better than the current best feasible solution. Note that this may speed up the search,
but may cause some solutions, including optimal ones, to be missed. If a model has been solved with a nonzero cheat value, then the optimal solution will be within the cheat value or less of the found solution. Observe that the option cheat is specified in absolute terms (like the option optCA), therefore non-negative values are appropriate for both minimization and maximization models. Note that using this option will invalidate any reporting of the dual bound or optimality gaps. Further, certain solver options can override the cheat setting, e.g., the Cplex option objDif, and some solvers may ignore the cheat option.

Default: 0

checkErrorLevel (boolean): Check errorLevel automatically after executing external program

Available: Command line, Option statement

If this option is set to 1, the errorLevel is checked implicitly after executing an external program via execute, put_utility exec and put_utility shell. An execution error is triggered and the execution is aborted, if it is not 0. So, with checkErrorLevel = 1 the before mentioned statements behave like execute.checkErrorLevel, put_utility exec.checkErrorLevel and put_utility shell.checkErrorLevel respectively.

Also, if this option is set as a command line parameter it initializes the state of the dollar control option $onerrorLevel. So, with checkErrorLevel = 1, the errorLevel is checked implicitly after $call and $hiddenCall. A compilation error is triggered and the compilation is aborted if that is not 0. So, $call behaves like $call.checkErrorLevel and $hiddenCall behaves like $hiddenCall.checkErrorLevel in this case.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not check errorLevel automatically after execution of external program</td>
</tr>
<tr>
<td>1</td>
<td>Check errorLevel automatically after execution of external program</td>
</tr>
</tbody>
</table>

CNS (string): Constrained Nonlinear Systems - default solver

Available: Command line, Option statement

The default solver for models of the type Constrained Nonlinear Systems is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

connectIn (string): Specify YAML Connect script file processed at start of GAMS

Available: Command line

The YAML file provided by this command line parameter is passed to and processed by Connect. This allows Connect to prepare data for import at the beginning of a GAMS job. The GAMS program itself does not need to know where the various data files (CSV, Excel, etc) are located. The supply side of the data contract is encapsulated in the YAML script.

Here is an example. The content of the file p.yaml that is supplied via the option connectIn could look like this:
PandasExcelReader:
   file: myworkbook.xlsx
   symbols:
      - name: p
        range: Sheet1!A1
        rowDimension: 1
        columnDimension: 1

GDXWriter:
   file: input.gdx
   symbols:
      - name: p

The GDX file input.gdx can be loaded at the beginning of a GAMS job using gdxLoad:

Set i, j;
Parameter p(i<,j<);
$gdxLoad input.gdx p
display p;

Alternatively, this can be nicely combined with the IDCGDXInput to load external input symbols. The GDX file input.gdx can be specified as value for IDCGDXInput (gams mymodel
   connectIn=p.yaml IDCGDXInput=input.gdx):

Set i, j;
$onExternalInput
Parameter p(i<,j<) / i1.j1 1 /;
$offExternalInput
display p;

The YAML code in the connectIn file can also utilize the GAMSReader agent. Normally, there
are no symbols to read at the job's start, but in case the job restarts from an existing work file
(see command line option restart) the symbols available from the work file can be read into
the Connect database with the GAMSReader agent. The YAML code in the connectIn file can
utilize the GAMSWriter agent. It will update the symbols and data read from the restart file
at the beginning of the GAMS compilation phase.

connectOut (string): Specify YAML Connect script file processed at end of GAMS

Available: Command line

The YAML file provided by this command line parameter is passed to and processed by
Connect. This allows Connect to take the GAMS data at the end of the GAMS job and export
it to various formats (CSV, Excel, etc) according to the instructions in the YAML file.

Here is an example. The content of the file p.yaml that is supplied via the option connectOut
could look like this:

- GDXReader:
   file: output.gdx
   symbols:
      - name: p

- PandasExcelWriter:
   file: myworkbook.xlsx
   symbols:
      - name: p
        range: Sheet1!A1
        rowDimension: 1

The GDX file output.gdx can be written at the end of a GAMS job using execute_unload:
Set i /i1*i10/, j /j1*j10/;
Parameter p(i,j); p(i,j) = uniform(0,1);
execute_unload 'output.gdx', p;

Alternatively, this can be nicely combined with the IDCGDXOutput to process external output symbols. The GDX file output.gdx can be specified as value for IDCGDXOutput (gams mymodel connectOut=p.yaml IDCGDXOutput=output.gdx):

Set i /i1*i10/, j /j1*j10/;
$onExternalOutput
Parameter p(i,j); p(i,j) = uniform(0,1);
$offExternalOutput

The YAML code in the connectOut file can also utilize the GAMSReader agent. This reads directly from GAMS memory (without the detour via GDX) and processes any symbol available at the end of the GAMS run. The YAML code in the connectOut file cannot utilize the GAMSWriter agent.

curDir (string): Current directory

Synonym: CDir
Available: Command line

This option sets the current working directory. It is useful when GAMS is called from an external system like Visual Basic. If it is not specified, it will be set to the directory the GAMS module is called from.

cutOff (real): Cutoff value for branch and bound

Available: Attribute statement (use before solve, reset by solve statement)

Within a branch-and-bound based solver, the parts of the tree with an objective value worse than the cutoff value are ignored. Note that this may speed up the initial phase of the branch and bound algorithm (before the first integer solution is found). However, the true optimum may be beyond the cutoff value. In this case the true optimum will be missed and moreover, no solution will be found.

Observe that this option is specified in absolute terms (like the option optCA).

Default: 0

decimals (integer): Decimal places for display statements

Available: Option statement

This option specifies the number of decimals that will be printed for numeric values that do not have a specific print format attached. The range is $[0, 8]$.

Default: 3

decryptKey (string): Key to decrypt a text file that was encrypted via $encrypt$

Available: Command line

This option provides a key to decrypt a GAMS input file that has been encrypted with a key provided by encryptKey and $encrypt$. For more information, see Encrypting Files.

defPoint (integer): Indicator for passing on default point

Available: Attribute statement (use before solve, reset by solve statement)

This option determines the point that is passed to the solver as a basis. By default, the levels and marginals from the current basis are passed to the solver. In some circumstances (mostly during debugging), it can be useful to pass a standard default input point, i.e. with all levels set to 0 or lower bound.
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pass user defined levels and marginals to solver</td>
</tr>
<tr>
<td>1</td>
<td>Pass default levels and marginals to solver</td>
</tr>
<tr>
<td>2</td>
<td>Pass default marginals to solver</td>
</tr>
</tbody>
</table>

**dFormat** *(integer)*: Date format

Synonym: DF

Available: Command line

This option controls the date format in the listing file. The three date formats correspond to various conventions used around the world. For example, the date December 2, 1996 will be written as 12/02/96 with the default df value of 0, as 02.12.96 with df=1, and as 96-12-02 with df=2.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Date as mm/dd/yy</td>
</tr>
<tr>
<td>1</td>
<td>Date as dd.mm.yy</td>
</tr>
<tr>
<td>2</td>
<td>Date as yy-mm-dy</td>
</tr>
</tbody>
</table>

**dictFile** *(real)*: Force writing of a dictionary file if dictfile > 0

Available: Attribute statement (use before solve, reset by solve statement)

If this option is set to a value that is larger than zero, it will instruct GAMS to make the GAMS names of variables and equations that have been generated by the solve statement available to the solver. In many solver links these names are registered with the solver and hence messages from the solver that involve variables and equations (e.g. an infeasible row or duplicate columns) can be easily interpreted by the user. Consider the following example:

Row 'demand(new-york)' infeasible, all entries at implied bounds.

Duplicate columns x(san-diego.new-york) and x(san-diego.chicago) make problem unbounded.

If we have modelname.dictfile=0 the same messages will read as follows:

Row 'c4' infeasible, all entries at implied bounds.

Duplicate columns x4 and x5 make problem unbounded.

Sometimes a dictionary is required for a successful run. Some solver option use the original GAMS names and need to be matched with the variables 1..n and equations 1..m in the solver. The dictionary file with its API allows to calculate such a mapping. Note that this is done automatically inside the solver links, so users do not need to be concerned with it.

For example, in the indicator constraints implementation a binary indicator variable is matched to a constraint. In the model [INDIC01] from the GAMS test library, this matching is done in the following GAMS/Cplex option file cplex.opt:

```plaintext
indic eq3(dice,f,fp)$comp(dice,f,fp) 1
```
Observe that if no dictionary is available, we will get an error:

**** Unable to read dictionary file required for indicator constraints

However, the dictionary comes at a price. Generating the names and calculating and storing the map takes time and space. In addition, GAMS names take up space in the solver. Thus, if the user needs very fast generation and does not need names, setting dictFile to zero is a good option.

Further, note that some solvers allow to suppress the loading of names (using the solver option names=no). Suppressing the loading of names facilitates to use the name mapping features required for models like INDIC01 above, but does not load the names into the solver name space for better reporting (and hence saves some space).

digit (string): Switch default for "$on/offDigit"

Available: Command line

For more info see $on/offDigit.

Default: off

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offDigit</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onDigit</td>
</tr>
</tbody>
</table>

dispWidth (integer): Number of characters to be printed in the column labels of all subsequent display statements

Available: Option statement

This option controls the number of characters that are shown for a label in a column in the context of the display statement. Consider the following example:

```gams
Set i / thislabelhas24characters /
Parameter p(i,i) / thislabelhas24characters.thislabelhas24characters 2/
display p;

option dispWidth=24;
display p;
```

The two display statements in this code will generate the following output:

```plaintext
---- 3 PARAMETER p
        thislabel~
thislabelhas24characters 2.000
---- 6 PARAMETER p
        thislabelhas24characters
thislabelhas24characters 2.000
```
Observe that in the first display, the label in the column is cut off after 10 characters, while in the second display it is shown in full.

Note that the default value is 10 and the range is $[10,31]$.

Default: 10

**dmpOpt (no value):** Debugging option: causes GAMS to echo the runtime option settings

Available: Option statement

This debugging option has the effect that all available option statements and their current values are listed in the listing file.

**dmpSym (no value):** Debugging option: causes GAMS to echo the symbol table to the listing file

Available: Option statement

This debugging option is especially useful for diagnosing memory problems. It has the effect that GAMS will report the number of elements that are stored for each identifier at the point in the program where this option is inserted together with a memory estimate. The report that is generated in this way is called a memory dump. This is similar to the option dmpUserSym, but prints GAMS internal symbols and information as well. For details, see section Finding the Causes for Excessive Memory Use.

**dmpUserSym (no value):** Debugging option: causes GAMS to echo the symbol table to the listing file for user defined symbols only

Available: Option statement

This debugging option is especially useful for diagnosing memory problems. It has the effect that GAMS will report the number of elements that are stored for each identifier at the point in the program where this option is inserted together with a memory estimate. The report that is generated in this way is called a memory dump. This is similar to the option dmpSym, but prints user defined symbols only and also leaves out some very technical information which are mostly for internal use. For details, see section Finding the Causes for Excessive Memory Use.

**DNLP (string):** Non-Linear Programming with Discontinuous Derivatives - default solver

Available: Command line, Option statement

The default solver for models of the type Nonlinear Programs with Discontinuous Derivatives is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

**docFile (string):** Filename stem for documentation files

Available: Command line

**domLim (integer):** Domain violation limit solver default
Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the maximum number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run and returns solver status 5 EVALUATION ERROR LIMIT. Nonlinear solvers have difficulties recovering after attempting an undefined operation. Note that some solvers operate in a mode where trial function evaluations are performed. These solvers will not move to points at which evaluation errors occur, thus the evaluation errors at trial points are not counted against the limit.

Default: ∞

**domUsd (integer):** Number of domain violations

Available: Attribute statement (use after solve)

This model attribute returns the number of domain violations after a solve.

**dualCheck (integer):** Output on the reduced cost condition

Available: Option statement

If this option is set to 1, the reduced cost condition for each variable in the column listing will be evaluated using the equation marginals. The default value is zero, which means that the calculation will be omitted.

Default: 0

**dumpOpt (integer):** Writes preprocessed input to the file input.dmp

Available: Command line

This option with value larger than 9 creates a GAMS input file of that will reproduce results encapsulating all include files into one GAMS file. If activated, a file will be written containing GAMS source code for the entire problem. The file name is the input file name with the extension dmp. For values smaller than 10, this option tries to encapsulate all the items from a restart file that are needed to execute a solve.

For the values smaller than 10, consider the following example. We will split the transportation model [TRNSPORT] into two files and run them with the save and restart feature. Then we will illustrate the option dumpOpt. The first file called trans1.gms contains the first part of the model up to and including the model statement:

Sets

\[
\text{i canning plants / seattle, san-diego /} \\
\text{j markets / new-york, chicago, topeka / ;}
\]

Parameters

\[
\text{a(i) capacity of plant i in cases} \\
\text{\quad / seattle 350, san-diego 600 /} \\
\text{b(j) demand at market j in cases} \\
\text{\quad / new-york 325, chicago 300, topeka 275 / ;}
\]

Table d(i,j) distance in thousands of miles

\[
\begin{array}{ccc}
\text{new-york} & \text{chicago} & \text{topeka} \\
\end{array}
\]
Scalar \( f \) freight in dollars per case per thousand miles /90/ ;

Parameter \( c(i,j) \) transport cost in thousands of dollars per case;
\[ c(i,j) = f \times d(i,j) / 1000 \]

Variables
- \( x(i,j) \) shipment quantities in cases
- \( z \) total transportation costs in thousands of dollars;

Positive Variable \( x \);

Equations
- \( \text{cost} \) define objective function
- \( \text{supply}(i) \) observe supply limit at plant \( i \)
- \( \text{demand}(j) \) satisfy demand at market \( j \);

\[
\begin{align*}
\text{cost} & \quad \text{define objective function} \\
\text{supply}(i) & \quad \text{observe supply limit at plant } i \\
\text{demand}(j) & \quad \text{satisfy demand at market } j \\
\end{align*}
\]

\[
\begin{align*}
\text{cost} & \quad z = \text{sum}((i,j), c(i,j) \times x(i,j)) \\
\text{supply}(i) & \quad \text{sum}(j, x(i,j)) = \text{a}(i) \\
\text{demand}(j) & \quad \text{sum}(i, x(i,j)) = \text{g} = b(j) \\
\end{align*}
\]

model transport /all/ ;

Note that we removed all comments for brevity. The second file called trans2.gms contains the solve statement and the display statement:

\[
\begin{align*}
\text{solve transport using lp minimizing } z \\
\text{display } x.l, x.m \\
\end{align*}
\]

We run the first file with the command line parameter save and thus generate a work file. For details on work files, see chapter The Save and Restart Feature. Then we run trans2.gms restarting from the saved work file. The result will be equivalent to running the original model [TRNSPORT].

Note
- The option dumpOpt can only be used effectively, if the first line in the second file, trans2.gms, is the solve statement.

Now, we will illustrate the use of the option dumpopt, by running the second file with the following command:

\[
> \text{gams} \text{ trans2 r=trans dumpopt=1}
\]

Here trans is the name of the saved files generated from the file trans1.gms. As a result of this call, a new file will be created. It is called trans2.dmp and has the following content:

* This file was written with DUMPOPT=1 at 11/25/21 15:15:17
* INPUT = C:\tmp\trans2.gms
* DUMP = C:\tmp\trans2.dmp
* RESTART = C:\tmp\0.g0?
* with time stamp of 11/16/21 17:18:53
* You may have to edit this file and the input file.
* There are 5 labels
Set WorkFileLabelOrder dummy set to establish the proper order /
   seattle,san-diego,new-york,chicago,topeka /;

Model transport;

Variable z 'total transportation costs in thousands of dollars';

Set i(*) 'canning plants' /
   seattle,san-diego /;

Set j(*) 'markets' /
   new-york,chicago,topeka /;

Parameter c(i,j) 'transport cost in thousands of dollars per case' /
   seattle.new-york 0.225,seattle.chicago 0.153,seattle.topeka 0.162,san-diego.new-york 0.225,san-diego.chicago 0.162,san-diego.topeka 0.126 /;

Positive Variable x(i,j) 'shipment quantities in cases';

Parameter a(i) 'capacity of plant i in cases' /
   seattle 350,san-diego 600 /;

Parameter b(j) 'demand at market j in cases' /
   new-york 325,chicago 300,topeka 275 /;

Equation demand(j) 'satisfy demand at market j';

Equation supply(i) 'observe supply limit at plant i';

Equation cost 'define objective function';

*     *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***

Note that all the data that enters the model in the solve statement has been regenerated. Observe that the parameter d has not been regenerated since it does not appear in the model. Changing the value of the parameter dumpopt will have the effect that other names are used for the identifiers in the regenerated file, see table below.

Note

If $onVerbatim is active, DumpOpt = 11 behaves like DumpOpt = 21 (comments are kept)

See also dumpOptGDX.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No dumpfile</td>
</tr>
<tr>
<td>1</td>
<td>Extract referenced data from the restart file using original set element names</td>
</tr>
<tr>
<td>2</td>
<td>Extract referenced data from the restart file using new set element names</td>
</tr>
<tr>
<td>3</td>
<td>Extract referenced data from the restart file using new set element names and drop symbol text</td>
</tr>
<tr>
<td>4</td>
<td>Extract referenced symbol declarations from the restart file</td>
</tr>
<tr>
<td>11</td>
<td>Write processed input file without comments</td>
</tr>
<tr>
<td>21</td>
<td>Write processed input file with all comments</td>
</tr>
</tbody>
</table>
dumpOptGDX (string): Defines a GDX file name stem created when using DumpOpt

Available: Command line

This parameter works together with the dumpOpt parameter. If that is set to 1, 11 or 21 while dumpOptGDX is set, GAMS will create a GDX file with the data used in the dump file instead of data statements, and the dump file will load the data from that GDX file. So, with the parameter dumpOptGDX=trns2 the example above would generate the following dump file:

* This file was written with DUMPOPT=1 at 11/25/21 15:14:11
* INPUT = C:\tmp\trans2.gms
* DUMP = C:\tmp\trans2.dmp
* RESTART = C:\tmp\0.g0?
* DUMPOPTGDX = trans2
* with time stamp of 11/16/21 17:18:53

$gdxIn trans2.gdx

* You may have to edit this file and the input file.

* There are 5 labels

Set WorkFileLabelOrder dummy set to establish the proper order;
$loadDCR WorkFileLabelOrder=* Model transport;

Variable z 'total transportation costs in thousands of dollars';

Set i(*) 'canning plants';
$loadDCR i

Set j(*) 'markets';
$loadDCR j

Parameter c(i,j) 'transport cost in thousands of dollars per case';
$loadDCR c

Positive Variable x(i,j) 'shipment quantities in cases';

Parameter a(i) 'capacity of plant i in cases';
$loadDCR a

Parameter b(j) 'demand at market j in cases';
$loadDCR b

Equation demand(j) 'satisfy demand at market j';

Equation supply(i) 'observe supply limit at plant i';

Equation cost 'define objective function';

$gdxIn

* *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
Note that more than one GDX file could be created with a common file stem defined by this parameter. This will be necessary if the source model has multiple data definitions for the same symbol like in this simple example:

Set  i / i1*i3/;
Parameter a(i) / i1 1, i2 2/;
$onMulti
Parameter a(i) / i1 0, i3 3/;
Scalar  b / 7/;
Scalar  b / 77/;

Running this as

> gams dummy dumpOpt=11 dumpOptGDX=dummyIn

will generate this dump file:

* This file was written with DUMPOPT=11 at 11/25/21 15:26:02
* * DUMP = C:\tmp\dummy.dmp
* * INPUT = C:\tmp\dummy.gms
* * RESTART =
* * DUMPOPTGDX = dummyIn
* *

$gdxIn dummyIn.gdx

Set  i  
$loadDCR i
  ;
Parameter a(i)  
$loadDCR a  
  ;
$onMulti
Parameter a(i)  
$gdxIn dummyIn_m1.gdx
$loadDCR a  
  ;
Scalar  b  
$loadDCR b  
  ;
Scalar  b  
$gdxIn dummyIn_m2.gdx
$loadDCR b  
  ;
* *** EXIT  C:\Data\gspTest\dummy.gms

$gdxIn
*** END OF DUMP ***

And this is the data found in the GDX files:

dummyIn.gdx:
Set  i(*) /  
 'i1',  
 'i2',  
 'i3' /;
Parameter a(*) /  
 'i1' 1,  
 'i2' 2 /;
dummyIn_m1.gdx:

Parameter a(*) /
  'i2' 2,
  'i3' 3 /;

Scalar b / 7 /;

dummyIn_m2.gdx:

Scalar b / 77 /;

dumpParms \text{(integer)}: GAMS parameter logging

Synonym: DP

Available: Command line

This option lists the settings of all command line parameters that were changed or set by the user, GAMS or an IDE during the current run. Note that with \text{dp=2} all file operations are listed, including the full path of each file on which any operation is performed.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No logging</td>
</tr>
<tr>
<td>1</td>
<td>Lists accepted/set parameters</td>
</tr>
<tr>
<td>2</td>
<td>Log of file operations plus list of accepted/set parameters</td>
</tr>
</tbody>
</table>

dumpParmsLogPrefix \text{(string)}: Prefix of lines triggered by DumpParms>1

Synonym: DPLP

Available: Command line

This option prefixes in the log file the list of all command line parameters that were changed or set by the user, GAMS or an IDE during the current run. Note that the option dumpParms must be greater than 1 for dumpParmsLogPrefix to have an effect.

Default: ***

ECImplicitLoad \text{(string)}: Allow implicit loading of symbols from embedded code or not

Available: Command line, Option statement

The command line parameter ECImplicitLoad initializes both, the option ECImplicitLoad and the dollar control option \$on/offECImplicitLoad for the compile-time equivalent behavior.

Default: on

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Do not allow implicit loading from embedded code</td>
</tr>
<tr>
<td>on</td>
<td>Allow implicit loading from embedded code</td>
</tr>
</tbody>
</table>
eject (no value): Inject a page break into the LST file

   Available: Option statement

   This option has the effect that a page break is inserted in the listing file.

EMP (string): Extended Mathematical Programs - default solver

   Available: Command line, Option statement

   The default solver for models of the type Extended Mathematical Programs is set during
   installation. The user may change this default by setting this option to the desired solver.

   Observe that if the solver was changed using an option statement, the default solver may be
   reset later in the program with another option statement, where the value of the option is set
   to default.

empty (string): Switch default for "$on/offEmpty"

   Available: Command line

   For more info see $on/offEmpty.

   Default: on

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offEmpty</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onEmpty</td>
</tr>
</tbody>
</table>

encryptKey (string): Key to encrypt a text file using $encrypt

   Available: Command line

   This option provides a key to encrypt a text file via $encrypt. For more information, see
   Encrypting Files.

eolCom (string): Switch default for "$on/offEolCom" and "$eolCom"

   Available: Command line

   If this is set to on or off it sets the state of $on/offEOLCom. Other strings (with not
   more than two characters) will set $EOLCom.

   Default: off

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offEolCom</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onEolCom using default EOL comment character</td>
</tr>
<tr>
<td>other</td>
<td>Activate $onEolCom setting specific EOL comment character(s)</td>
</tr>
</tbody>
</table>

eolOnly (integer): Single key-value pairs (immediate switch)

   Synonym: EY
Available: Command line

This option controls formatting of parameters on the command line and is useful in conjunction with the option `parmFile`.

This option acts as an immediate switch that forces only one option-value pair to be read on a line. Note that by default, any number of option-value pairs may be present on the same line and termination characters and quoting is necessary to determine the end of key/value pair. With this option active the remainder after the key is used as the value independent of quoting or termination characters.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Any number of keys or values</td>
</tr>
<tr>
<td>1</td>
<td>Only one key-value pair on a line</td>
</tr>
</tbody>
</table>

**epsToZero (string):** Treat eps as zero when unloading to GDX

Available: Command line, Option statement

When this is set to `on`, EPS values are written as zero when unloading parameters or variable and equation levels to GDX at execution time. The command line parameter `EpsToZero` initializes both, the option `EpsToZero` and the dollar control option `$on/offEpsToZero` for the compile-time equivalent behavior.

Default: `off`

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Treat Eps as Eps</td>
</tr>
<tr>
<td>on</td>
<td>Treat Eps as Zero</td>
</tr>
</tbody>
</table>

**errMsg (integer):** Placing of compilation error messages

Available: Command line

This option controls the position of the compilation error messages in the listing file. To illustrate the option, consider the following slice of GAMS code:

```gams
Set i / 1*10 /;
Set j(i) / 10*11 /;
Parameter a(jj) / 12 25.0 /;
```

After running this code, the listing file will contain the following lines:

```
1 Set i / 1*10 /;
2 Set j(i) / 10*11 /;
**** $170
3 Parameter a(jj) / 12 25.0 /;
**** $120
4
120 Unknown identifier entered as set
170 Domain violation for element
```

** **** 2 ERROR(S) 0 WARNING(S)**
Note that numbers $170$ and $120$ flag the two errors as they occur, but the errors are explained only at the end of the compilation output. However, if the code is run using the option `errmsg=1`, the resulting listing file will contain the following:

```plaintext
1 Set i / 1*10 / ;
2 Set j(i) / 10*11 / ;
**** $170
**** 170 Domain violation for element
3 Parameter a(jj) / 12 25.0 / ;
**** $120
**** 120 Unknown identifier entered as set
4
**** 2 ERROR(S) 0 WARNING(S)
```

Observe that the explanation for each error is provided immediately following the error marker.

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Place error messages at the end of compiler listing</td>
</tr>
<tr>
<td>1</td>
<td>Place error messages immediately following the line with the error</td>
</tr>
<tr>
<td>2</td>
<td>Suppress error messages</td>
</tr>
</tbody>
</table>

**errNam** *(string)*: Name of error message file

Available: Command line

This option specifies the name of a file defining the internally used compiler error messages. It is used to change the name from the default name `gamserrs.txt`.

**error** *(string)*: Force a compilation error with message

Available: Command line

This option forces a parameter error with a specified message. It is useful in the context of incorporating a GAMS file within another batch file where the user needs to have control over the conditions when GAMS is called. See also section Conditional Compilation.

To illustrate, the default GAMS log file from running a model with the option `error=Hallo` will look as follows:

```plaintext
gams: **** Error: Parameter error(s)
      : Reading parameter(s) from "command line"
      : *** Error Hallo
      : Finished reading from "command line"
```

**errorLog** *(integer)*: Max error message lines written to the log for each error

Synonym: ER

Available: Command line

This option controls the number of error message lines that are written to the log file.

Under GAMS Studio, the default is reset to 99.

Default: 2147483647
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error messages to LOG file</td>
</tr>
<tr>
<td>n</td>
<td>Number of lines for each error that will be written to LOG file</td>
</tr>
</tbody>
</table>

**etAlg** *(real)*: Solver dependent timing information

Available: Attribute statement (use after solve)

Unlike **etSolve** and **etSolver**, this attribute is set by the individual solver links. If not set, it defaults to **NA**. This attribute was intended to allow solvers to return the elapsed time used by the solve algorithm without including any model generation, communication, or setup time. However, solvers are free to adapt this convention and return time-related information (but not necessarily elapsed time) for executing the solve algorithm. Please inspect your solver manual for the actual meaning of the value returned in this attribute.

**etLim** *(real)*: Elapsed time limit in seconds

Synonym: ETL

Available: Command line

This option controls the time limit for a GAMS job. The system will terminate with a compilation or execution error if the limit is reached. A GAMS job will terminate if the elapsed time in seconds exceeds the value of **etLim**. If a solve statement gets executed, and the **resLim** for the model to be solved is greater than **etLim - timeElapsed**, **resLim** will be reduced automatically.

Default: $\infty$

**etSolve** *(real)*: Elapsed time it took to execute a solve statement in total

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time it took to execute a solve statement in total. This time includes the model generation time, the time to read and write files, the time to create the solution report and the time taken by the actual solve. The time is expressed in seconds of wall-clock time.

**etSolver** *(real)*: Elapsed time taken by the solver only

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time taken by the solver only. This does not include the GAMS model generation time and the time taken to report and load the solution back into the GAMS database. The time is expressed in seconds of wall-clock time.

**execMode** *(integer)*: Limits on external programs that are allowed to be executed

Available: Command line

A higher value denotes a more restrictive alternative. If the restriction level **n** is chosen, then the restriction levels less than **n** will also be active. For example, if restriction level 3 is chosen, then restrictions 2 and 1 will apply too.

Default: 0
### 4.40 The GAMS Call and Command Line Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Everything allowed</td>
</tr>
<tr>
<td>1</td>
<td>Interactive shells in $call and execute commands are prohibited</td>
</tr>
<tr>
<td>2</td>
<td>Embedded Code and all $call and execute commands are prohibited</td>
</tr>
<tr>
<td>3</td>
<td>$echo or put commands can only write to directories in or below the working or scratchdir</td>
</tr>
<tr>
<td>4</td>
<td>$echo and put commands are not allowed</td>
</tr>
</tbody>
</table>

**expand** *(string)*: Expanded (include) input file name

Synonym: EF

Available: Command line

This option generates a file that contains information about all the input files processed during a particular compilation. By default, the names of the input files are composed by completing the name with the current directory.

Consider the following example:

```
$call rm expfile.txt
$onecho > file1.inc
a = a*2; display a;
$include file2.inc
$offecho
$onecho > file2.inc
a = a+1; display a;
$include file3.inc
$offecho
$onecho > file3.inc
a = a**2; display a;
$offecho
parameter a / 1 /;
$include file3.inc
$include file2.inc
$include file1.inc
```

If the model is run with the command line parameter **ef=expfile.txt**, a file called `expfile.txt` will be generated. This file will contain the following lines:

```
1 INPUT 0 0 0 1 29 C:\GAMS\Examples\expand.gms
2 CALL 0 1 1 1 1 1 rm expfile.txt
3 INCLUDE 1 1 14 14 15 C:\GAMS\Examples\file3.inc
4 INCLUDE 1 1 15 16 19 C:\GAMS\Examples\file2.inc
5 INCLUDE 2 4 2 18 19 C:\GAMS\Examples\file3.inc
6 INCLUDE 1 1 16 20 25 C:\GAMS\Examples\file1.inc
7 INCLUDE 2 6 2 22 25 C:\GAMS\Examples\file2.inc
8 INCLUDE 3 7 2 24 25 C:\GAMS\Examples\file3.inc
9 EXIT 0 1 20 29 29 C:\GAMS\Examples\expand.gms
```

Note that the first row always refers to the parent file, in this case the file `expand.gms`. The first column gives the sequence number of the input files that were encountered. The second column refers to the type of file that is referenced. The following file types are possible:
Observe that $\textit{call}$ is also listed. The third column describes the depth for nested include files. The fourth column provides the sequence number of the parent file for the file being referenced. The fifth column gives the local line number in the parent file where the dollar control option $\textit{include}$ appeared. The sixth column gives the global (expanded) line number which contained $\textit{include}$. The seventh column provides the total number of lines in the file after it is processed. The last column provides the name of the file.

Note that the listing in the expand file is similar to the include file summary in the listing file of the model. And like the include file summary, this file will not be written, if $\textit{offInclude}$ is set in the model.

\textbf{fdDelta (real): Step size for finite differences}

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control the step size while the numerical Hessian and numerical derivatives are computed in the context of the \texttt{function} suffixes \texttt{.hessn} and \texttt{.gradn}. For functions with one argument, GAMS evaluates the function at $f(x-d)$ and $f(x+d)$ for the numerical gradient. If function values are used for the numerical Hessian, GAMS will evaluate at $f(x-2d)$, $f(x)$ and $f(x+2d)$. For functions with multiple arguments, the same calculations are performed for the components of the input argument vector.

Default: $1.0\times10^{-5}$

\textbf{fdOpt (integer): Options for finite differences}

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control how numerical derivatives are computed. The values provide choice regarding the scaling of steps, Hessian calculation method and the use of numerical first derivatives.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All derivatives analytically, for numerical Hessian use gradient values, scale delta</td>
</tr>
<tr>
<td>1</td>
<td>All derivatives analytically, for numerical Hessian use function values, scale delta</td>
</tr>
<tr>
<td>2</td>
<td>Gradient analytically, force Hessian numerically using gradient values, scale delta</td>
</tr>
<tr>
<td>3</td>
<td>Gradient analytically, force Hessian numerically using function values, scale delta</td>
</tr>
<tr>
<td>4</td>
<td>Force gradient and Hessian numerically, scale delta</td>
</tr>
<tr>
<td>10</td>
<td>Same as 0, but no scale of delta</td>
</tr>
<tr>
<td>11</td>
<td>Same as 1, but no scale of delta</td>
</tr>
<tr>
<td>12</td>
<td>Same as 2, but no scale of delta</td>
</tr>
<tr>
<td>13</td>
<td>Same as 3, but no scale of delta</td>
</tr>
<tr>
<td>14</td>
<td>Same as 4, but no scale of delta</td>
</tr>
</tbody>
</table>

\textbf{fErr (string): Alternative error message file}
Available: Command line

This option redirects the compilation error messages to a file and names the file. By default, the file name is composed by completing the name with the scratch directory and the scratch extension. Note that under default settings such a file with compilation error messages is not generated. This option can be used when GAMS is being integrated into other environments like Visual Basic. The error messages that are reported in the listing file may be extracted with this option and their display may be controlled from the environment that is calling GAMS.

To illustrate, consider the slice of GAMS code that we used to explain the option errMsg. If we call this code with the command line parameter `ferr=myfile.err`, a file called `myfile.err` will be created in the scratch directory. This file will contain the following lines:

```
0 0 0 0 D:\GAMS\NEW.LST
1 1 170 31 D:\GAMS\NEW.GMS
2 2 120 14 D:\GAMS\NEW.GMS
```

Note that the first column refers to the global row number of the error in the listing file. The second column refers to the row number of the error in the individual file where the problem occurred. This will be different from the first column only if the error occurred in an include file. In this case, the second column will contain the line number in the include file where the error occurred, while the first number will contain the global line number (as reported in the listing file) where the error occurred. The number in the third column refers to the error number of the compilation error. The fourth number refers to the column number of the error in the source file. The last column contains the individual file in which the error occurred.

**fileCase (integer):** Casing of file names and paths (put, gdx, ref, $include, etc.)

Available: Command line

This option facilitates modifying the case of file names and paths. It applies to files created by a GAMS Job such as for example put files, GDX files and reference files. Under Windows, the casing of a created file will only be affected if the file does not yet exist, i.e. `fileCase=2` won’t create `transport.ref` if there is already a file `TRANSPORT.ref`. It should be noted that `fileCase` also applies to existing files that are used but not created by a GAMS job such as for example `$include files` or `$batInclude files`.

Note that many other file names and paths are affected by this option (e.g. the scratch directory (`scrDir`) or the GAMS system directory (`sysDir`)) and that it is recommended to use this option with caution.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Causes GAMS to use default casing</td>
</tr>
<tr>
<td>1</td>
<td>Causes GAMS to upper case file names including path of the file</td>
</tr>
<tr>
<td>2</td>
<td>Causes GAMS to lower case file names including path of the file</td>
</tr>
<tr>
<td>3</td>
<td>Causes GAMS to upper case file names only (leave the path alone)</td>
</tr>
<tr>
<td>4</td>
<td>Causes GAMS to lower case file names only (leave the path alone)</td>
</tr>
</tbody>
</table>

**fileStem (string):** Sets the file stem for output files which use the input file name as stem by default

Available: Command line

By default, some output files use the input file name as base. If the names of these output files were not set explicitly, then this option may be used to set another name than the input
file name as base for these output files. In particular, the names for the following files may be set with fileStem: dump files (see option dumpOpt), GDX files (if the option GDX was set to default), log files (see option logFile), lst files (see option output), reference files (if the option reference was set to default) and trace summary files (see also option trace).

fileStemApFromEnv (string): Append a string read from an environment variable to the "FileStem"

Available: Command line

This option for users that submit GAMS job via mpirun/mpiexec. Such commands will spawn multiple instances of GAMS (the precise number is an argument to mpirun/mpiexec). Each invocation of GAMS will run the identical job and only the contents of an environment variable (PMI_RANK) will differentiate the run. Since GAMS will normally write to modelname.log/lst if we run the GAMS file modelname.gms we will have many jobs writing to the same file. Therefore we use this option to append the content of a particular environment variable (name given by this option) to the default file names (see fileStem). Hence GAMS will create modelname0.log/lst, modelname1.log/lst, and so forth when started with mpirun/mpiexec and fileStemApFromEnv is set to the environment variable that provides the MPI rank of the invocation. We allow to specify the name of the environment variable because different MPI implementations use different variable names (e.g. PMI_RANK or OMPI_COMM_WORLD_RANK).

filtered (string): Switch default for "$on/offFiltered"

Available: Command line

For more info see $on/offFiltered.

Default: on

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offFiltered</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onFiltered</td>
</tr>
</tbody>
</table>

forceOptFile (integer): Overwrites other option file section mechanism

Available: Command line

Default: 0

forceWork (boolean): Force GAMS to process a save file created with a newer GAMS version or with execution errors

Synonym: FW

Available: Command line

Work files generated by GAMS using the command line parameter save are saved in binary format. The information inside these files will change from one GAMS version to another GAMS version. GAMS makes every attempt to be backward compatible and ensure that all new GAMS systems are able to read save files generated by older GAMS systems. However, this cannot be guaranteed the other way around. Thus, GAMS does not allow to process a save file, that was generated by a newer version of GAMS in general. This can be changed by setting forceWork to 1. Also, GAMS does not continue after processing a save while with an execution error by default. This behavior can be changed as well by setting forceWork=1.
Attention

Forcing GAMS to continue after reading a save file from a newer GAMS version or a save file with execution errors may lead to unexpected behavior. If it is about the GAMS version and you can recreate the save file you may write it using GAMS command line parameter `previousWork=1`.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No translation</td>
</tr>
<tr>
<td>1</td>
<td>Try translation</td>
</tr>
</tbody>
</table>

**forLim** (*integer*): GAMS looping limit

Available: Command line, Option statement

This option specifies the maximum number of permitted executions of control structures with a `for` statement, a `while` statement or a `repeat` statement before GAMS signals an execution error and terminates the control structure.

Default: \(\infty\)

**fSave** (*boolean*): Creates a forced work file, i.e., the file is saved even if execution errors or other errors occurred

Available: Command line

This option allows to save a file even in the face of execution or other errors. How it works depends on the command line parameter `save`.

Note that the option value of 1 is mainly used by solvers that can be interrupted from the terminal.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Workfile only written to file specified by SAVE if no errors occur</td>
</tr>
<tr>
<td>1</td>
<td>Workfile always written to file specified by SAVE or if SAVE is not present to a name made up by GAMS</td>
</tr>
</tbody>
</table>

**G205** (*integer*): Use GAMS version 2.05 syntax

Available: Command line

This option sets the level of the GAMS syntax and is mainly used to ensure backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when they are run with the latest version of GAMS. This option enables users to run such old models.

For example, the word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting the option `g205=1` allows the word "if" to be used as an identifier since it was not a keyword in Release 2.05. As another example, the word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting the option `g205=2` allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.
Note

If the values 1 or 2 are specified for option g205, then it will not be permitted to use enhancements of the GAMS language that were introduced in later versions.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use only latest syntax</td>
</tr>
<tr>
<td>1</td>
<td>Allow version 2.05 syntax only</td>
</tr>
<tr>
<td>2</td>
<td>Allow version 2.25 syntax only</td>
</tr>
</tbody>
</table>

**GDX** *(string)*: GAMS data exchange file name

Available: Command line

This option specifies the name of the GAMS data exchange file and causes a GDX file to be written that contains all data in the model at the end of the job. Setting `gdx` to the string 'default' causes GAMS to create a GDX file with the gms file root name and a gdx extension. Thus, e.g., the call

```
> gams trnsport gdx=default
```

will cause GAMS to write the GDX file `trnsport.gdx`.

**gdxCompress** *(boolean)*: Compression of generated GDX file

Available: Command line

This option specifies whether the GDX files are compressed or not.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not compress GDX files</td>
</tr>
<tr>
<td>1</td>
<td>Compress GDX files</td>
</tr>
</tbody>
</table>

**gdxConvert** *(string)*: Version of GDX files generated (for backward compatibility)

Available: Command line

This option specifies in which format the GDX files will be written.

Default: v7

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>v5</td>
<td>Version 5 GDX file, does not support compression</td>
</tr>
<tr>
<td>v6</td>
<td>Version 6 GDX file</td>
</tr>
<tr>
<td>v7</td>
<td>Version 7 GDX file</td>
</tr>
</tbody>
</table>

**gdxUels** *(string)*: Unload labels or UELs to GDX either squeezed or full

Available: Command line, Option statement
This option specifies the UEL export mode. The UEL table may be written to a GDX file in two different modes. In **squeezed** mode, only the UELs that are required by the exported symbols are exported. In **full** mode, all UELs are exported. The following code snippet illustrates the difference:

```gams
Set i / i1*i5 /;
Parameter p(i) / i3 3 /;
option gdxuels = squeezed;
execute_unload 'squeezed' p;
execute 'gdxdump squeezed UelTable=i';

option gdxuels = full;
execute_unload 'full' p;
execute 'gdxdump full UelTable=i';
```

The file `squeezed.gdx` will contain the following lines:

```plaintext
Set i /
   'i3' /;

Parameter p(*) /
   'i3' 3 /;
```

The file `full.gdx` on the other hand, will contain the following lines:

```plaintext
Set i /
   'i1' ,
   'i2' ,
   'i3' ,
   'i4' ,
   'i5' /;

Parameter p(*) /
   'i3' 3 /;
```

Default: **squeezed**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squeezed</td>
<td>Write only the UELs to Universe, that are used by the exported symbols</td>
</tr>
<tr>
<td>Full</td>
<td>Write all UELs to Universe</td>
</tr>
</tbody>
</table>

**gridDir** *(string)*: Grid file directory

Synonym: GDir

Available: Command line

This option sets the grid file directory. Note that each GAMS job has only one grid file directory.

**gridScript** *(string)*: Grid submission script

Synonym: GScript

Available: Command line
This option provides the name of a script file that is used to submit grid computing jobs. If only the file name is given the file is assumed to be located in the system directory. A fully qualified name can be given as well. The script needs to be similar to the file \texttt{gmsgrid.cmd} on Windows machines with arguments that give name and location of the solver executable, the solver control file name and the name of the scratch directory. For an example of such a script, see section \textit{The Grid Facility: Architecture and Customization}. However, note that advanced knowledge of how GAMS sets up and calls solvers is needed for successful use.

Default: \texttt{gmsgrid}

\textbf{handle (real):} Unique handle number of SOLVE statement

Available: Attribute statement (use after solve)

The model attribute \texttt{handle} contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The handle number may be used by the functions \texttt{handleCollect}, \texttt{handleStatus}, \texttt{handleDelete}, \texttt{handleSubmit} and \texttt{readyCollect}. For details see chapter \textit{The Grid and Multi-Threading Solve Facility}.

\textbf{heapLimit (real):} Maximum Heap size allowed in MB

Synonym: \texttt{HL}

Available: Command line

This option allows to limit the amount of memory a GAMS job may use during compilation and execution. If the needed data storage exceeds this limit, the job will be terminated.

Default: $\infty$

\textbf{holdFixed (boolean):} Treat fixed variables as constants

Available: Command line, Attribute statement (use before solve)

This option facilitates treating fixed variables as constants. Thus the problems size may be reduced.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Fixed variables are not treated as constants</td>
</tr>
<tr>
<td>1</td>
<td>Fixed variables are treated as constants</td>
</tr>
</tbody>
</table>

\textbf{holdFixedAsync (boolean):} Allow HoldFixed for models solved asynchronously as well

Available: Command line, Option statement

By default, \texttt{holdFixed} is automatically deactivated if a model is solved asynchronously, since this could lead to inconsistent solutions otherwise. To allow this anyway, this parameter can be set to 1.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore HoldFixed setting for async solves</td>
</tr>
<tr>
<td>1</td>
<td>Allow HoldFixed for async solves</td>
</tr>
</tbody>
</table>
**IDCGDXInput** *(string)*: GDX file name with data for implicit input  
Available: Command line  
Specify the name of a GDX file that is used for implicit loading of input data. Details about this are described with `$onExternalInput`.  

**IDCGDXOutput** *(string)*: GDX file name for data for implicit output  
Available: Command line  
Specify the name of a GDX file that is used for implicit writing of output data. Details about this are described with `$onExternalOutput`.  

**IDCGenerateGDX** *(string)*: Specify GDX file name of input and output side of data contract  
Available: Command line  
Specify the name of a GDX file that is used to store all symbols that are declared as external input or external output at the end of a GAMS run.  

**IDCGenerateGDXInput** *(string)*: Specify GDX file name of input side of data contract  
Available: Command line  
Specify the name of a GDX file that is used to store all symbols that are declared as external input at the end of a GAMS run.  

**IDCGenerateGDXOutput** *(string)*: Specify GDX file name of output side of data contract  
Available: Command line  
Specify the name of a GDX file that is used to store all symbols that are declared as external output at the end of a GAMS run.  

**IDCGenerateJSON** *(string)*: Specify JSON file name of data contract  
Available: Command line  
Specify the name of a JSON file that is used to store the information (but not the data) about all symbols that are declared as external input or external output at the end of a GAMS run.  

**IDCJSON** *(string)*: Specify JSON file name to verify data contract  
Available: Command line  
Specify the name of a JSON file that is read and used to verify that the implicit data contract given by the GAMS model by declaring symbols as external input or external output matches the expected symbol information given by this specified file. This is the expected schema of that file:  
```json  
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Input/Output configuration file schema",
  "description": "Configuration file is automatically generated by GAMS and describes the data contract.",
  "type": "object",
  "definitions": {
    "symbolHeader": {
      "description": "Symbol header",
      "type": "object",
      "properties": {
        "alias": {
          "description": "Explanatory text of the header",
          "type": "string",
          "minLength": 1
        }
      }
    }
  }
}  ```
},
  "type":{
    "description":"GAMS input type",
    "type":"string",
    "enum":{
      "string",
      "numeric"
    }
  }},
  "required": ["alias", "type"]}
},
"symbol":{
  "description": "Symbol definition",
  "type": "object",
  "properties": {
    "alias":{
      "description": "Explanatory text of the symbol",
      "type": "string",
      "minLength":1
    },
    "symtype":{
      "description": "Symbol type",
      "type": "string",
      "default": "en",
      "enum":{
        "set",
        "parameter",
        "variable",
        "equation"
      }
    },
    "headers":{
      "description": "Symbol headers",
      "type": "object",
      "minProperties":1,
      "additionalProperties":{
        "$ref": "#/definitions/symbolHeader"
      }
    }
  },
  "required": ["alias", "symtype", "headers"]
},
"scalarSymbol": {
  "description": "Scalar symbols that are grouped together in one table",
  "type": "object",
  "properties": {
    "alias":{
      "description": "Explanatory text of the symbol",
      "type": "string",
      "minLength":1
    },
    "symnames":{
      "description": "Symbol names in scalar tables",
      "type": "array",
      "minItems":1,
      "uniqueItems":true,
      "items":{
        "type": "string",
        "minLength":1
      }
    },
    "symtypes":{
      "description": "Symbol types in scalar tables",
      "type": "array",
      "minItems":1,
      "items":{
        "type": "string",
        "enum":{
          "set",
          "parameter",
          "variable",
          "equation"
        },
        "minLength":1
      }
    },
    "symtext":{
      "description": "Symbol aliases (exp. text) in scalar tables",
      "type": "array",
      "minItems":1,
      "items":{
        "type": "string",
        "minLength":1
      }
    }
  },
  "headers":{
4.40 The GAMS Call and Command Line Parameters

IDCProtect (boolean): Flag to control assignment protection of external input symbols

Available: Command line, Option statement

By default, it is not allowed to change symbols which are declared as external input at execution time. This parameter allows to ignore this restriction. It sets the initial state for the dollar control options $on/offIDCProtect.

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Allow to change external input symbols at execution time</td>
</tr>
<tr>
<td>1</td>
<td>Protect external input symbols from being changed at execution time</td>
</tr>
</tbody>
</table>

IDE (boolean): Integrated Development Environment flag

Available: Command line

This option instructs GAMS to write special instructions to the log file that are in turn read by an IDE.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown environment</td>
</tr>
<tr>
<td>1</td>
<td>Runs under GAMS IDE</td>
</tr>
</tbody>
</table>
**inlineCom (string):** Switch default for "$on/offInline" and "$inlineCom"

Available: Command line

If this is set to on or off it sets the state of $on/offInline. Other strings (a pair with not more than two characters each) will set $inlineCom.

Default: off

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offInline</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onInline using default inline comment characters</td>
</tr>
<tr>
<td>other</td>
<td>Activate $onInline setting specific inline comment characters</td>
</tr>
</tbody>
</table>

**input (string):** Input file

Synonym: I

Available: Command line

Completing the input file name with the current directory composes the final name. If such a file does not exist and the extension was not specified, the standard input extension will be attached and a second attempt will be made to open an input file.

**inputDir, inputDir1..40 (string):** Input file directories (searched at compile time)

Synonym: IDIR

Available: Command line

In general, GAMS searches for input and include files in the current working directory only. This option allows the user to specify additional directories for GAMS to search for include and batinclude files as well as GDX file via $gdxIn. A maximum of 40 separate directories may be included. The directories are separated by Operating System specific symbols. For example, on a PC the separator is a semicolon (;) character and under Unix it is the colon (:) character. Note that libinclude and sysinclude files are handled differently. Their paths are specified with the command line parameters libIncDir and sysIncDir respectively.

Consider the following illustration:

> gams myfile idir \mydir;\mydir2

Note that the search order for the file myfile.gms and all included files in PC systems will be as follows: (1) the current directory, (2) the directories specified by inputdir in their respective order (here the directories: \mydir and \mydir2). Under Unix, the corresponding GAMS call will be:

> gams myfile idir \mydir:\mydir2

Note that the information in the option inputDir may be also transferred to GAMS by entering the individual directories separately. A maximum of 40 directories may be passed on in this manner. The number appended to InputDir is important since the earlier InputDir directories will be searched first.

The example above may alternatively be formulated in the following way:

> gams myfile idir1 mydir1 idir2 mydir2
Note that in this case the search order will be as follows:

1. current directory
2. mydir1
3. mydir2

Observe that we could modify the command in the following way:

> gams myfile idir3 \mydir1 idir2 \mydir2

Note that in this case the search order will be as follows:

1. current directory
2. mydir2
3. mydir3

Thus it is not the order in which the directories are specified that matters, but the number of the option `inputDir` that they have been assigned to.

**integer1..5 (integer)**: Integer communication cell N

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an integer communication cell that may contain any integer number.

**interactiveSolver (boolean)**: Allow solver to interact via command line input

Available: Command line

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interaction with solvelink 0 is not supported</td>
</tr>
<tr>
<td>1</td>
<td>Interaction with solvelink 0 is supported</td>
</tr>
</tbody>
</table>

**intVarUp (integer)**: Set mode for default upper bounds on integer variables

Available: Command line, Option statement

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Set default upper bound for integer variables to +INF</td>
</tr>
<tr>
<td>1</td>
<td>Pass a value of 100 instead of +INF to the solver as upper bound for integer variables</td>
</tr>
<tr>
<td>2</td>
<td>Same as 0 but writes a message to the log if the level of an integer variable is greater than 100</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2 but issues an execution error if the level of an integer variable is greater than 100</td>
</tr>
</tbody>
</table>

**iterLim (integer)**: Iteration limit of solver
Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the maximum number of permitted solver iterations, before the solver terminates the run. If this limit is reached, the solver will terminate and will return solver status 2 ITERATION INTERRUPT. Note that the definition of what constitutes an iteration depends on the solver. For LP solvers, \textit{iterlim} often refers to the number of simplex iterations (i.e., pivots). For MIP solvers, \textit{iterlim} often refers to the cumulative number of simplex iterations over all solves of LP relaxations. Observe that \textit{iterlim} does not apply to all iterations. For example, it might not apply to barrier iterations and major iterations in nonlinear solvers. For these iterations solver-specific options need to be set. Moreover, some solver links might \textit{reinterpret} the value of this option. For example, if left at default, the solver link might use the default iteration limit of the solver.

Default: 2147483647

\textbf{iterUsd (integer): Number of iterations used}

Available: Attribute statement (use after solve)

This model attribute returns the number of iterations used after a solve.

\textbf{jobTrace (string): Job trace string to be written to the trace file at the end of a GAMS job}

Synonym: JT

Available: Command line

This option specifies a string that is written to the trace file at the end of a GAMS job.

\textbf{keep (boolean): Controls keeping or deletion of process directory and scratch files}

Available: Command line

This option controls whether to keep the process directory. In the process directory the temporary/scratch files are located, unless the options \textit{scrDir} or \textit{procDir} were used.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Delete process directory</td>
</tr>
<tr>
<td>1</td>
<td>Keep process directory</td>
</tr>
</tbody>
</table>

\textbf{libIncDir (string): LibInclude directory}

Synonym: LDIR

Available: Command line

This option specifies the name of the directory to be used by GAMS for $\textit{libInclude}$ files that do not have a full path specification. An absolute or relative path may be specified. If the option \textit{libIncDir} is not set, it will be set to the subdirectory \textit{inclib} in the GAMS standard locations. A relative path is relative to the GAMS system directory.

Note that if the option \textit{libIncDir} parameter is set, the default library include directories will only be searched, if the libInclude file cannot be found in the specified folder.
4.40 The GAMS Call and Command Line Parameters

Attention

Only one directory may be set with the option libIncDir. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

> gams myfile libIncDir mydir

Note that GAMS will search for any referenced libInclude file in the directory `<GAMS System Directory>/mydir` first.

license (string): Use alternative license file

Available: Command line

This option specifies the name the file that contains the GAMS license. It may be used to point to an explicit license file rather letting GAMS search for the default `gamslice.txt` in various locations including the GAMS system directory. The locations ("Data directories") can be printed by running `gamsinst -listdirs`, see installation notes for details. The license option should only be used by advanced users attempting to override internal license information.

limCol (integer): Maximum number of columns listed in one variable block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of columns that are listed for each variable in the column listing in the listing file. Note that the value of zero will suppress the column listing.

Default: 3

limRow (integer): Maximum number of rows listed in one equation block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of rows that are listed for each equation in the equation listing in the listing file. Note that the value of zero will suppress the equation listing.

Default: 3

line (integer): Line number of last solve of the corresponding model

Available: Attribute statement (use after solve)

This model attribute returns the line number of the last solve of the respective model.

linkUsed (integer): Integer number that indicates the value of SolveLink used for the last solve

Available: Attribute statement (use after solve)

This model attribute returns an integer number that indicates the value of the option solveLink that was used for the last solve.

listing (string): Switch default for "$on/offListing"

Available: Command line

For more info see $on/offListing.

Default: on
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offListing</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onListing</td>
</tr>
</tbody>
</table>

**logFile (string):** Log file name

Synonym: LF

Available: Command line

This option is used in conjunction with the option logOption or short LO. If lo is set to 2 or 4, this option will specify the name of the log file name. The name provided by the option is completed using the current directory. If no log file is given, but the value of lo is 2 or 4, then the file name will be the input file name with the extension .log.

Consider the following GAMS call:

```plaintext
> gams trnsport lo=2 lf=myfile.log
```

Note that the resulting log file will redirected to the file myfile.log. It will contain the following lines:

--- Starting compilation
--- trnsport.gms(75) 3 Mb
--- Starting execution: elapsed 0:00:00.017
--- trnsport.gms(44) 4 Mb
--- Generating LP model transport
--- trnsport.gms(63) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.072

IBM ILOG CPLEX 24.8.2 r59988 Released Jan 3, 2017 DEG x86 64bit/Mac OS X
Cplex 12.7.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Dual Objective</th>
<th>In Variable</th>
<th>Out Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73.125000</td>
<td>x(Seattle.New-York)</td>
<td>demand(New-York) slack</td>
</tr>
<tr>
<td>2</td>
<td>119.025000</td>
<td>x(Seattle.Chicago)</td>
<td>demand(Chicago) slack</td>
</tr>
<tr>
<td>3</td>
<td>153.675000</td>
<td>x(San-Diego.Topeka)</td>
<td>demand(Topeka) slack</td>
</tr>
<tr>
<td>4</td>
<td>153.675000</td>
<td>x(San-Diego.New-York)</td>
<td>supply(Seattle) slack</td>
</tr>
</tbody>
</table>

LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective : 153.675000

--- Restarting execution
--- trnsport.gms(63) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(72) 3 Mb
instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 02/09/17 14:39:20 elapsed 0:00:00.280

**logLine** (integer): Amount of line tracing to the log file

Synonym: LL

Available: Command line

This option may be used to limit the number of line tracing sent out to the log file during the compilation phase of a GAMS run. Note that setting this option to zero will cause the line tracing to be suppressed for all phases of the GAMS processing.

The log file that results from running the model [TRNSPORT] with the option ll=0 is shown below.

--- Starting compilation
--- Starting execution: elapsed 0:00:00.018
--- Generating LP model transport
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.060

IBM ILOG CPLEX 24.8.2 r59988 Released Jan 3, 2017 DEG x86 64bit/Mac OS X
Cplex 12.7.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Dual Objective</th>
<th>In Variable</th>
<th>Out Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73.125000</td>
<td>x(Seattle.New-York)</td>
<td>demand(New-York)</td>
</tr>
<tr>
<td>2</td>
<td>119.025000</td>
<td>x(Seattle.Chicago)</td>
<td>demand(Chicago)</td>
</tr>
<tr>
<td>3</td>
<td>153.675000</td>
<td>x(San-Diego.Topeka)</td>
<td>demand(Topeka)</td>
</tr>
<tr>
<td>4</td>
<td>153.675000</td>
<td>x(San-Diego.New-York)</td>
<td>supply(Seattle)</td>
</tr>
</tbody>
</table>

LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective : 153.675000

--- Restarting execution
--- Reading solution for model transport
instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 02/09/17 15:43:43 elapsed 0:00:00.275

If we compare this output to the output shown in the example of option logFile, we will observe that the line numbers are missing from this log file.

Default: 2
logOption (integer): Log option

Synonym: LO

Available: Command line

This option controls the location of the output log of a GAMS run. By default, GAMS directs the log of the run to the screen/console. If logOption is set to 2, the log will be redirected to a file. With logOption=3 all the output goes to the standard output. Note that if no file name is provided for the log through the option logFile, the file name will be the input file name with the extension .log. Observe that the settings zero and 2 may be used to permit jobs to run in the background.

To illustrate, consider the following call:

```plaintext
> gams trnsport lo=2
```

Note that the resulting log file, trnsport.log, will be identical to the file myfile.log that is shown as part of the description of the option logFile.

Default: 3

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No log output</td>
</tr>
<tr>
<td>1</td>
<td>Log output to screen (console)</td>
</tr>
<tr>
<td>2</td>
<td>Log output to logfile</td>
</tr>
<tr>
<td>3</td>
<td>Log output to standard output</td>
</tr>
<tr>
<td>4</td>
<td>Log output to logfile and standard output</td>
</tr>
</tbody>
</table>

LP (string): Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Linear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

lstTitleLeftAligned (boolean): Write title of LST file all left aligned

Available: Command line

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Split LST title into left and right aligned part</td>
</tr>
<tr>
<td>1</td>
<td>Write LST title completely left aligned</td>
</tr>
</tbody>
</table>
marginals (*integer*): Indicator for marginals present

Available: Attribute statement (use after solve)

**maxExecError** (*integer*): Execution time error limit

Synonym: ExecErr

Available: Command line

This option puts an upper limit on the number of errors that may be found during execution or preprocessing associated with a solve statement. If more than maxExecError errors have been found GAMS will abort. This value can be overwritten with the function maxExecError.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No errors allowed limit</td>
</tr>
<tr>
<td>n</td>
<td>Max number allowed</td>
</tr>
</tbody>
</table>

**maxGenericFiles** (*integer*): Maximum number of generic file names tried at execution time file creation

Available: Command line, Option statement

In case of a failed writing of put files and GDX files via execute_unload (e.g. because the given path is invalid or the file is open by another program), GAMS tries to write to a generic file name instead. The number of file names tried can be specified with this option. To avoid this completely and throw an error right away, if the given name is invalid, set this to 0.

Default: 20

**maxInfes** (*real*): Maximum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the maximum number of infeasibilities after a solve.

**maxProcDir** (*integer*): Maximum number of 225* process directories

Available: Command line

This option controls the maximum number of work file directories that may be generated by GAMS. By default they are called 225a, 225b, ... 225aa, 225ab ... Note that the label 225 may be changed with the option procDir.

Default: 700

**MCP** (*string*): Mixed Complementarity Problems - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Complementarity Problems is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.
**MCPRHoldFx (boolean):** Print list of rows that are perpendicular to variables removed due to the holdfixed setting

Available: Command line, Option statement, Attribute statement (use before solve)

If option holdfixed is true, fixed columns in row.column matches are removed from MCP models but the matching rows remain. These rows are ignored by the solver. This option causes a list of such rows to be included in the listing file prior to the solve summary.

Default: 0

**meanInfes (real):** Mean of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the mean of the infeasibilities after a solve.

**measure (no value):** Output of time and memory use since the last measure statement or the program beginning

Available: Option statement

This option has the effect that three measurements will be displayed: the memory and time usage since the last option statement measure and the total time used. See also the related option profile.

**memoryStat (no value):** Show memory statistics in the LST file

Available: Option statement

This option has the effect that memory statistics will be shown in the listing file.

**MINLP (string):** Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Nonlinear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

**MIP (string):** Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

**MIQCP (string):** Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

**modelStat (integer):** Integer number that indicates the model status

Available: Attribute statement (use after solve)

This model attribute returns the model status after a solve. Observe that there are compile-time constants that are related to modelStat. Note that additional information to the values given in the table below is provided in section Model Status.
The default solver for models of the type Mathematical Program with Equilibrium Constraints is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

multi (string): Switch default for "$on/offMulti[R]"

Available: Command line

If this is set to on or off it sets the state of $on/offMulti. The value onR activates $onMultiR.

Default: off

multiPass (boolean): Multipass facility

Synonym: MP

Available: Command line
This option may be used to instruct GAMS to use a quick syntax checking compilation facility which does not require all items to be declared. This alternative is useful when a large model is assembled from smaller pieces. It allows slices of GAMS code to be independently checked for syntax errors.

Consider the following example:

\[
\begin{align*}
   a(i) &= b(i)*5 ;
   b(i) &= c(j) ;
\end{align*}
\]

By default, running a file containing just these two statements will generate the following listing file:

\[
\begin{align*}
   1 & a(i) = b(i)*5 ; \\
   **** & $140,120,140 \\
   2 & b(i) = c(j) ; \\
   **** & $140,120,149 \\
   120 & \text{Unknown identifier entered as set} \\
   140 & \text{Unknown symbol} \\
   149 & \text{Uncontrolled set entered as constant} \\
   **** & 6 \text{ ERROR(S) 0 WARNING(S)}
\end{align*}
\]

Note that both sets i and j have not been defined or initialized. In addition, the identifiers a, b and c have not been defined either. Further, an assignment cannot be made without the right-hand side of the assignment being known. However, in both assignments there is no data available for the right-hand side. If we run the same two lines with the option \texttt{mp=1}, we will get the following listing file:

\[
\begin{align*}
   1 & a(i) = b(i)*5 ; \\
   2 & b(i) = c(j) ; \\
   **** & $149 \\
   149 & \text{Uncontrolled set entered as constant} \\
   **** & 1 \text{ ERROR(S) 0 WARNING(S)}
\end{align*}
\]

Observe that the statements have now been processed independently of their context. They are now checked only for consistency. GAMS now assumes that the sets i and j, as well as the identifiers a, b, and c have been defined and, if necessary, initialized elsewhere. The only error that is reported is the inconsistency of indices in the second statement.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard compilation</td>
</tr>
<tr>
<td>1</td>
<td>Check-out compilation</td>
</tr>
<tr>
<td>2</td>
<td>As 1, and skip $\text{call}$ and ignore missing file errors with $\text{include}$ and $\text{gdxin}$ as well as unknown dimension errors with empty data statements</td>
</tr>
</tbody>
</table>

\textbf{NLP (string)}: Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type \texttt{Nonlinear Programs} is set during installation. The user may change this default by setting this option to the desired solver.
Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

**nodLim (integer):** Node limit in branch and bound tree

Available: Command line, Attribute statement (use before solve)

This option specifies the maximum number of nodes that are to be processed in the branch and bound tree search for a MIP problem. Note that setting `nodLim` can stop solutions that are exhibiting “excessive” iterations: if the limit is reached the algorithm will terminate, without obtaining optimality. In this case the solver status will be 4 TERMINATED BY SOLVER. Observe that a value of zero is interpreted as 'not set'.

Default: 0

**nodUsd (integer):** Number of nodes used by the MIP solver

Available: Attribute statement (use after solve)

This model attribute returns the number of nodes used by the MIP solver after a solve.

**noNewVarEqu (integer):** Triggers a compilation error when new equations or variable symbols are introduced

Available: Command line

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AllowNewVarEqu</td>
</tr>
<tr>
<td>1</td>
<td>DoNotAllowNewVarEqu</td>
</tr>
</tbody>
</table>

**number (real):** Model instance serial number

Available: Attribute statement (use after solve)

This model attribute returns the model instance serial number. Note that the first model solved is assigned number 1, the second number 2 etc. The user may also set a value n and the next model solved will be assigned the number n+1.

**numDepnd (integer):** Number of dependencies in a CNS model

Available: Attribute statement (use after solve)

This model attribute returns the number of dependencies identified in a CNS model after a solve.

**numDVar (integer):** Number of discrete variables

Available: Attribute statement (use after solve)

This model attribute returns the number of discrete variables after a solve.

**numEqu (integer):** Number of equations
Available: Attribute statement (use after solve)
This model attribute returns the number of equations after a solve.

\texttt{numInfes} (integer): Number of infeasibilities
Available: Attribute statement (use after solve)
This model attribute returns the number of infeasibilities after a solve.

\texttt{numNLIns} (integer): Number of nonlinear instructions
Available: Attribute statement (use after solve)
This model attribute returns the number of nonlinear instructions after a solve.

\texttt{numNLNZ} (integer): Number of nonlinear nonzeros
Available: Attribute statement (use after solve)
This model attribute returns the number of nonlinear nonzeros after a solve.

\texttt{numNOpt} (integer): Number of nonoptimalities
Available: Attribute statement (use after solve)
This model attribute returns the number of nonoptimalities after a solve.

\texttt{numNZ} (integer): Number of nonzero entries in the model coefficient matrix
Available: Attribute statement (use after solve)
This model attribute returns the number of nonzero entries in the model coefficient matrix after a solve.

\texttt{numRedef} (integer): Number of MCP redefinitions
Available: Attribute statement (use after solve)
This model attribute returns the number of MCP equation-type redefinitions after a solve.

\texttt{numVar} (integer): Number of variables
Available: Attribute statement (use after solve)
This model attribute returns the number of variables after a solve.

\texttt{numVarProj} (integer): Number of bound projections during model generation
Available: Attribute statement (use after solve)
This model attribute returns the number of bound projections during model generation.

\texttt{objEst} (real): Estimate of the best possible solution for a mixed-integer model
Available: Attribute statement (use after solve)
This model attribute returns the estimate of the best possible solution for a MIP or other models with discrete variables. The model attribute is mainly used after solve.

Some GAMS solvers implement algorithms (e.g. branch-and-bound) that generate a bound on the objective function value for the best possible solution. Users may access this bound by using the model attribute \texttt{objEst}. Note that this is mainly used for models with discrete variables (e.g. MIP and MINLP), but some global solvers implement spatial branch-and-bound algorithms that also provide such a bound for purely continuous problems. In case the solver does not set the attribute, its value is \texttt{na}.

\texttt{objVal} (real): Objective function value
Available: Attribute statement (use after solve)
This model attribute returns the objective function value after a solve.

\texttt{on115} (boolean): Generate errors for unknown unique element in an equation
Available: Command line
This option generates errors for unknown unique elements in an equation.
Default: 0
4.40 The GAMS Call and Command Line Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error messages</td>
</tr>
<tr>
<td>1</td>
<td>Issue error messages</td>
</tr>
</tbody>
</table>

**optCA (real):** Absolute Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an *absolute termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The absolute gap is defined to be |PB-DB|. Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization).

If the absolute gap is not greater than optCA, the solver will terminate and return solver status 1 NORMAL COMPLETION and model status 8 INTEGER SOLUTION (for a problem with discrete variables) or 2 LOCAL OPTIMAL or 7 FEASIBLE SOLUTION (for a problem without discrete variables). Note that this is a termination test only; setting this option should not change the global search.

**Note**

As this is an *absolute* criterion, setting the value to 100 means that the objective value will be within the 100 units of the true objective value.

Observe that a nonzero value for optCA will reduce solution time. However, it may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by optCA of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: 0

**optCR (real):** Relative Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies a *relative termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The precise definition of optCR depends on the solver. GAMS and some solvers use the following formula to compute the *optimality gap*:

$$\frac{|PB-DB|}{\max(|PB|,|DB|)}$$

Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization). However, two other formulas are also widely used, namely

$$\frac{|PB-DB|}{|PB|}$$

and

$$\frac{|PB-DB|}{|DB|}$$

Different adjustments when the denominator approaches zero or bounds are of different signs will be applied. The solver will stop as soon as it has found a feasible solution proven to be within optCR of optimal, that is, the optimality gap falls below optCR.
Note

As optCR is specified in proportional terms relative to the objective value, a value of 0.1 means that the objective value will be within 10% of the true objective value.

Observe that the solver will stop after finding a solution proven to be optimal within the tolerance specified with optCR and thus the solution time may be reduced. However, setting this option may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by optCR of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: 1.0E-04

optDir (string): Option file directory

Available: Command line

This option may be used to specify the name of the directory for solver option files. By default, the directory will be set to the current working directory.

optFile (integer): Default option file

Available: Command line, Attribute statement (use before solve)

This option instructs the solver to read an option file. The value of optFile determines which option file is used (see table below). Solver options allow users to manipulate the way solvers work. This may affect various solver functions including the choice of the branch and bound tree handling strategies. Please consult the solver manuals for options for each solver.

Note that this option is available as model attribute and command line parameter. Consider the following GAMS call:

> gams myfile optfile=1

Observe that the value of 1 for optFile means that the option file with the name solverName.opt will be used. Here solverName is the name of the respective solver. For example, if the solver CONOPT is used, the name of the respective option file is conopt.opt.

Note

If optFile is set with the model attribute in the GAMS input file, the value of the model attribute will override any optFile specifications on the command line.

Different values for optFile allow access to different option files for the same solver. Note that the following rule is used: if we specify optfile = n, then solvername.opt will be used for n=1, otherwise solvername.opX, solvername.oXX or solvername.XXX will be used, where X's are the characters representing the value of n, for n > 1. Observe that no option file will be used if the value of optFile is zero.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No option file will be used</td>
</tr>
<tr>
<td>1</td>
<td>The option file solvername.opt will be used</td>
</tr>
<tr>
<td>2</td>
<td>The option file solvername.op2 will be used</td>
</tr>
<tr>
<td>3</td>
<td>The option file solvername.op3 will be used</td>
</tr>
<tr>
<td>15</td>
<td>The option file solvername.o15 will be used</td>
</tr>
<tr>
<td>222</td>
<td>The option file solvername.222 will be used</td>
</tr>
<tr>
<td>1234</td>
<td>The option file solvername.1234 will be used</td>
</tr>
</tbody>
</table>
output \textit{(string):} Listing file name

Synonym: OO

Available: Command line

By default, the name of the output file (or listing file) is automatically created by combining
the name of the input file with the current directory and applying the standard output file
extension \textit{.lst}. This option may be used to specify an alternative name for the output file. If
the value is a file name without an absolute path, the current directory will compose the final
name. If the absolute path is included in the file name, then the name is used as specified.

Consider the following examples:

\begin{verbatim}
gams trnsport
\end{verbatim}
\begin{verbatim}
gams trnsport o=trnsport.out
\end{verbatim}
\begin{verbatim}
gams trnsport o=c:\test\trnsport.out
\end{verbatim}

Note that the first call will create an output file called \textit{trnsport.lst} (for PC and Unix platforms) in the current directory. The second call will create a file called \textit{trnsport.out} in the current directory. The last call will create the file as listed. If the directory \texttt{c:\test} does not exist, GAMS will exit with a parameter error.

Creation of the lst file can be suppressed by setting \texttt{output} to \texttt{NUL} under Windows or to
\texttt{/dev/null} on most Unix machines.

\textbf{pageContr (integer):} Output file page control option

Synonym: PC

Available: Command line

This option affects the page control in the listing file.

Default: 2

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No page control, with padding</td>
</tr>
<tr>
<td>1</td>
<td>FORTRAN style line printer format</td>
</tr>
<tr>
<td>2</td>
<td>No page control, no padding</td>
</tr>
<tr>
<td>3</td>
<td>Formfeed character for new page</td>
</tr>
</tbody>
</table>

\textbf{pageSize (integer):} Output file page size (=0 no paging)

Synonym: PS

Available: Command line

This option specifies the number of lines that are used on a page for printing the listing file.
The lower bound is zero, which is interpreted as +inf. That means that everything is printed
to one page.

Default: 0

\textbf{pageWidth (integer):} Output file page width
Synonym: PW

Available: Command line

This option sets the print width on a page in the listing file with a possible range from 72 to 32767. If the value is outside the allowed range, the default value will be used.

Note that .pw is also a put file attribute. In the context of the put writing facility, it may be used to set the page width of a put file. See page width for further details.

Default: 32767

**parmFile** *(string): Command Line Parameter include file*

Synonym: PF

Available: Command line

This option specifies the name of a secondary customization parameter file to use. It is used to augment the command line adding more command line parameters from a file. It is read from the current directory unless a path is specified. For an example, see section Specifying Options Through a Secondary Parameter File.

**pLicense** *(string): Privacy license file name*

Available: Command line

This option gives the name of a privacy license file that contains file encryption codes. A full path should be used. For more information, see Encrypting Files.

**prefixLoadPath** *(boolean): Prepend GAMS system directory to library load path*

Available: Command line

The OS environment variable to locate shared libraries used to be prefixed with the GAMS system directory. The option controls if this done or not. For the platforms Windows and AIX setting of this option has no impact.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not set GAMS system directory at beginning of library load path</td>
</tr>
<tr>
<td>1</td>
<td>Set GAMS system directory at beginning of library load path</td>
</tr>
</tbody>
</table>

**previousWork** *(boolean): Indicator for writing workfile with previous workfile version*

Available: Command line

When GAMS creates a workfile (e.g. using command line parameter restart) by default it uses the most recent version of the workfile format. GAMS is backward compatible with respect to workfile formats, i.e. newer GAMS versions can process workfiles generated by older GAMS version. As long as the workfile format has not changed older GAMS versions can even process workfiles generated by newer GAMS versions (using the forceWork=1 command line parameter). This particular command line parameter allows to create a workfile using the previous workfile format with the current GAMS version, so the previous GAMS version...
(and probably a few more versions back) will properly restart from this workfile even without issuing a warning.

This option is mostly used in combination with submissions to the NEOS server. Setting this option to 1 allows to submit NEOS jobs with a restart file even if the NEOS version is somewhat older than the local GAMS version.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Write workfile using the current version</td>
</tr>
<tr>
<td>1</td>
<td>Write workfile using the previous workfile version</td>
</tr>
</tbody>
</table>

priorOpt *(real)*: Priority option for variable attribute .prior

Available: Attribute statement (use before solve)

Instructs the solver to use the priority branching information passed by GAMS through variable suffix values variable.prior. If and how priorities are used is solver-dependent.

Default: 0

procDir *(string)*: Process Directory

Available: Command line

This option specifies the name of the process directory. If specified, the directory must already exist and it will not be deleted when GAMS cleans up. By default, the process directory name is chosen automatically from the list 225a, 225b, ..., 225aa, 225ab ..., by skipping over existing entries, and the directory will be deleted during cleanup if the option keep is not used. Very little is written to the process directory, but the scratch directory is used more, and the option scrDir takes its default from the process directory.

This option specifies the directory where the process directory should be created. If specified, the directory must already exist. While the process directory does not get cleaned automatically if the option procDir is set, this is not the case if the option procDirPath is used instead. Thus it allows to conveniently change the location of the process directory without changing the GAMS cleanup behavior. Note that if the location of the process directory is changed, the location of the default scratch directory will be changed accordingly (see option scrDir).

procTreeMemMonitor *(boolean)*: Monitor the memory used by the GAMS process tree

Available: Command line

Setting this option to 1 will cause GAMS to record the high-memory mark for the GAMS process tree (i.e. the gams or gams.exe process and all its children) and note this information in the log just prior to finishing. This is done via a separate thread that runs periodically (use option procTreeMemTicks to control how often) to construct the GAMS process tree and compute the memory usage for each process in it. There is some overhead with this so it is suggested to use this option only when needed and with a procTreeMemTicks value not smaller than the default.

In addition to the memory high-water marks (measured using both the size of the resident set and the virtual set used by each process in the tree) the report indicates the number of times this measurement was taken and any failures during this computation.

Default: 0
**Value** | **Meaning**
---|---
0 | Do not monitor memory usage for the GAMS process tree
1 | Start a thread to monitor memory usage for the GAMS process tree

**procTreeMemTicks** *(integer)*: Set wait interval between memory monitor checks: ticks = milliseconds

Available: Command line

Sets the wait interval in milliseconds between checks of the GAMS process tree memory usage: see `procTreeMemMonitor` for details.

Default: 2000

**procUsed** *(integer)*: Integer number that indicates the used model type

Available: Attribute statement (use after solve)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LP</td>
</tr>
<tr>
<td>2</td>
<td>MIP</td>
</tr>
<tr>
<td>3</td>
<td>RMIP</td>
</tr>
<tr>
<td>4</td>
<td>NLP</td>
</tr>
<tr>
<td>5</td>
<td>MCP</td>
</tr>
<tr>
<td>6</td>
<td>MPEC</td>
</tr>
<tr>
<td>7</td>
<td>RMPEC</td>
</tr>
<tr>
<td>8</td>
<td>CNS</td>
</tr>
<tr>
<td>9</td>
<td>DNLP</td>
</tr>
<tr>
<td>10</td>
<td>RMINLP</td>
</tr>
<tr>
<td>11</td>
<td>MINLP</td>
</tr>
<tr>
<td>12</td>
<td>QCP</td>
</tr>
<tr>
<td>13</td>
<td>MIQCP</td>
</tr>
<tr>
<td>14</td>
<td>RMIQCP</td>
</tr>
<tr>
<td>15</td>
<td>EMP</td>
</tr>
</tbody>
</table>

**profile** *(integer)*: Execution profiling

Available: Command line, Option statement

The *execution profile* of a GAMS run contains the individual and cumulative time required to execute the sections of the GAMS model, as well as information on memory use. The option `profile` controls whether an execution profile will be generated in the listing file. Observe that `profile` is available as command line parameter and option statement.

Note

The value for `profile` that is specified with an option statement in the GAMS input file overrides the value of `profile` that is passed through the command line.

Observe that an execution profile will be generated if the option `profile` is assigned a value larger than zero (zero is the default). Setting `profile` to 1 has the effect that execution times for each statement and the number of set elements over which the particular statement is executed will be reported. However, statements in programming flow control structures like loops will be omitted. Information on the execution of these statements will be included in
the profile if the value is \( n \), with \( n > 1 \). Note that an overview of the values for profile is given in the table at the end of this description.

Consider the following GAMS call:

\[
> \text{gams transport profile=1}
\]

This call causes the following additional lines to appear in the listing file:

\[
\begin{align*}
- & - 1 \text{ InitE} & 0.000 & 0.000 \text{ SECS} & 3 \text{ MB} \\
- & - 1 \text{ ExecInit} & 0.000 & 0.000 \text{ SECS} & 3 \text{ MB} \\
- & - 44 \text{ Assignment c} & 0.011 & 0.011 \text{ SECS} & 4 \text{ MB} & 6 \\
- & - 63 \text{ Assignment transport} & 0.000 & 0.011 \text{ SECS} & 4 \text{ MB} & 3 \\
- & - 65 \text{ Solve Init transport} & 0.000 & 0.012 \text{ SECS} & 4 \text{ MB} \\
- & - 57 \text{ Equation cost} & 0.001 & 0.013 \text{ SECS} & 4 \text{ MB} & 1 \\
- & - 58 \text{ Equation supply} & 0.000 & 0.013 \text{ SECS} & 4 \text{ MB} & 2 \\
- & - 59 \text{ Equation demand} & 0.000 & 0.013 \text{ SECS} & 4 \text{ MB} & 3 \\
- & - 65 \text{ Solve Fini transport} & 0.009 & 0.022 \text{ SECS} & 4 \text{ MB} & 19 \\
- & - 65 \text{ GAMS Fini} & 0.001 & 0.001 \text{ SECS} & 4 \text{ MB} \\
- & - 1 \text{ InitE} & 0.000 & 0.000 \text{ SECS} & 2 \text{ MB} \\
- & - 1 \text{ ExecInit} & 0.000 & 0.000 \text{ SECS} & 2 \text{ MB} \\
- & - 65 \text{ Solve Alg transport} & 0.000 & 0.000 \text{ SECS} & 2 \text{ MB} \\
- & - 65 \text{ Solve Read transport} & 0.002 & 0.002 \text{ SECS} & 2 \text{ MB} \\
- & - 67 \text{ Display} & 0.000 & 0.002 \text{ SECS} & 3 \text{ MB} \\
- & - 69 \text{ Display} & 0.000 & 0.002 \text{ SECS} & 3 \text{ MB} \\
- & - 69 \text{ GAMS Fini} & 0.001 & 0.001 \text{ SECS} & 3 \text{ MB}
\end{align*}
\]

Observe that the first column provides the line number in the input file of the statement that is executed.

The second column reports the type of the respective statement. For an overview of all GAMS statements, see section Classification of GAMS Statements. In addition, ExecInit denotes the beginning of the execution phase of the GAMS input file and GAMS Fini denotes the end of this phase. Note that as soon as a solve statement is processed, GAMS will pass control to the solver system. Once the solver has completed its task, GAMS will restart. Thus we have two ExecInit/GAMS Fini pairs in our example. Note that only equations are listed, and not variables. This reflects the fact that GAMS uses an equation based scheme to generate a model.

The third and fourth columns show the individual time needed to execute the statement and the cumulative time taken by the GAMS system so far.

The last column gives the number of assignments that were generated in the specified line.

In addition to the lines above, a profile summary is created at the end of the listing file. This summary contains (up to) ten of the slowest execution steps. The profile summary from transport.lst follows:

\[
\begin{align*}
\text{----- Profile Summary (17 records processed)} \\
& 0.011 & 0.004\text{GB} & 44 \text{ Assignment c (6)} \\
& 0.009 & 0.004\text{GB} & 65 \text{ Solve Fini transport (19)} \\
& 0.002 & 0.002\text{GB} & 65 \text{ Solve Read transport} \\
& 0.001 & 0.004\text{GB} & 65 \text{ GAMS Fini} \\
& 0.001 & 0.004\text{GB} & 57 \text{ Equation cost (1)} \\
& 0.001 & 0.003\text{GB} & 69 \text{ GAMS Fini}
\end{align*}
\]

Note that execution profiles and profile summaries are particularly useful for detecting the sources of performance problems. For further details, see section Finding the Causes for Slow Program Execution.

Default: 0
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No profiling</td>
</tr>
<tr>
<td>1</td>
<td>Minimum profiling</td>
</tr>
<tr>
<td>n</td>
<td>Profiling depth for nested control structures</td>
</tr>
</tbody>
</table>

**profileFile** *(string)*: Write profile information to this file

Synonym: PFILE

Available: Command line

This option causes profiling information to be written to a file. Note that profiling information is only created with the setting `profile=1` or `profile=2`. For example such a file may have the following content:

```
1   -1   0.000   0.003 ExecInit
45   6   0.000   0.004 Assignment c
66   -1   0.000   0.004 Solve Init transport
58    1   0.000   0.004 Equation cost
60    2   0.000   0.004 Equation supply
62    3   0.000   0.004 Equation demand
66    19   0.015   0.004 Solve Fini transport
66    -1   0.000   0.004 GAMS Fini
  1   -1   0.000   0.002 ExecInit
  66   -1   0.000   0.002 Solve Read transport
  68   -1   0.000   0.003 Display
  68    -1   0.000   0.003 GAMS Fini
```

**profileTol** *(real)*: Minimum time a statement must use to appear in profile generated output

Synonym: PTOL

Available: Command line, Option statement

This option sets the profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file. Note that this option is only effective if the value of the option `profile` is larger than zero.

Default: 0

**putDir** *(string)*: Put file directory

Synonym: PDir

Available: Command line

By default, put files are generated and saved in the current working directory. This option may be used to specify an alternative directory. Note that this option does not work if an absolute file name is provided through the `file` statement.

**putND** *(integer)*: Number of decimals for put files

Available: Command line

This sets the default for the put file attribute `.nd`.

Default: 2

**putNR** *(integer)*: Numeric round format for put files

Available: Command line

This sets the default for the put file attribute `.nr`.

Default: 1
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Item is displayed in F or E format</td>
</tr>
<tr>
<td>1</td>
<td>Item is rounded to fit given width and decimals</td>
</tr>
<tr>
<td>2</td>
<td>Item is displayed in scientific notation</td>
</tr>
<tr>
<td>3</td>
<td>Item is rounded to fit given width</td>
</tr>
<tr>
<td>4</td>
<td>Item is displayed in F or E format ignoring given decimals</td>
</tr>
</tbody>
</table>

**putPS** (*integer*): Page size for put files

Available: Command line

This sets the default for the put file attribute `.ps`.

Default: 58

**putPW** (*integer*): Page width for put files

Available: Command line

This sets the default for the put file attribute `.pw`.

Default: 32767

**pyMultInst** (*integer*): GAMS/Python Multiple Instance Interpreter

Available: Command line

This option controls the setup of the Python interpreter for use in embedded Python code sections. If this option is set to 1 the embedded Python code can work with multiple independent Python sessions. If `pyMultInst` is set to 0 there is a single interpreter even with multiple parallel embedded Python sessions that all share the same data space. With `pyMultInst` set to 0 or 1, the environment variable `GMSPYTHONMULTINST` is set by GAMS to this value. If the option is not set the value will be taken from the user environment variable `GMSPYTHONMULTINST`. Note that setting this to 1 might cause problems when using third party modules and packages (e.g. `numpy` or modules that make use of it) and might also impact the performance.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use a single-session Python interpreter</td>
</tr>
<tr>
<td>1</td>
<td>Use a multi-session Python interpreter</td>
</tr>
</tbody>
</table>

**QCP** (*string*): Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`. 
real1..5 (real): Real communication cell N

Available: Option statement, Attribute statement (use before solve)

This option specifies a real communication cell that may contain any real number.

reference (string): Symbol reference file

Synonym: RF

Available: Command line

If this option is specified, all symbol references will be written to the specified file. Setting rf or Reference to the string 'default' will cause GAMS to create a reference file with the file root name of the GAMS input file and the extension ref. Thus the call

`> gams trnsport rf=default`

will generate the reference file `trnsport.ref`.

referenceLineNo (string): Controls the line numbers written to a reference file

Synonym: RFLN

Available: Command line

Default: actual

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual</td>
<td>Actual line number of symbol reference</td>
</tr>
<tr>
<td>start</td>
<td>Line number where the statement with the reference starts</td>
</tr>
</tbody>
</table>

reform (integer): Reformulation level

Available: Option statement, Attribute statement (use before solve)

This option triggers an objective function reformulation. The interpretation depends on the solver. The solvers MINOS and SNOPT support this option. Note that the default value is zero and the range is [-2147483647,2147483647].

resCalc (real): Time spent in function and derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resDeriv (real): Time spent in derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resGen (real): Time GAMS took to generate the model in CPU seconds (deprecated)

Available: Attribute statement (use after solve)

This model attribute returns the time GAMS took to generate the model in CPU seconds.

resIn (real): Time to import model (deprecated)
4.40 The GAMS Call and Command Line Parameters

Available: Attribute statement (use after solve)

**resLim** *(real):* Wall-clock time limit for solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the time in seconds that the solver can run before it can terminate and return the solver status 3 **RESOURCE INTERRUPT**. The solver should start the clock soon after it starts, so the time required to read in the problem and do any reformulations, preprocessing or presolving is included in the time limit. Where possible, the time limit applies to the wall-clock time: this behavior translates well to multi-threaded solves. Moreover, some solver links might *reinterpret* the value of this option. For example, if left at default (1e+10), the solver link might use the default time limit of the solver.

Note

This value could be automatically reduced, if required because of **etLim**.

Default: 10000000000

**resOut** *(real):* Time to export solution (deprecated)

Available: Attribute statement (use after solve)

**restart** *(string):* Name of a restart file, see **The Save and Restart Feature**

Synonym: R

Available: Command line

This option specifies the name of a work file that was written with the option **save** that will be used to restart the GAMS program. The work file is also called **restart file**. For more information including examples, see chapter **The Save and Restart Feature**.

**restartNamed** *(string):* Name of another matching restart file, see **Obfuscated Work Files**

Synonym: RN

Available: Command line

**resUsd** *(real):* Time the solver used to solve the model in seconds

Available: Attribute statement (use after solve)

This model attribute returns the time in seconds used by the solver. Wherever possible, the units used (wall-clock time vs. CPU time) will be the same as used by the **reslim** option.

**RMINLP** *(string):* Relaxed Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type **Relaxed Mixed Integer Nonlinear Programs** is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.
RMIP (string): Relaxed Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mixed Integer Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

RMIQCP (string): Relaxed Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mixed Integer Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

RMPEC (string): Relaxed Mathematical Programs with Equilibrium Constraints - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mathematical Program with Equilibrium Constraints is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to default.

rngBndMax (real): Maximum absolute non-zero value of bounds passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

rngBndMin (real): Minimum absolute non-zero value of bounds passed to the solver

Available: Attribute statement (use after solve)

rngMatMax (real): Maximum absolute non-zero value of coefficients in the model matrix passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

Note
For non-linear models, the value of the coefficients looked at depends on the activity levels of one or more of the variables. More details are described here.

rngMatMin (real): Minimum absolute non-zero value of coefficients in the model matrix passed to the solver

Available: Attribute statement (use after solve)
Note
For non-linear models, the value of the coefficients looked at depends on the activity levels of one or more of the variables. More details are described here.

rngRhsMax (real): Maximum absolute non-zero value of right hand sides passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

rngRhsMin (real): Minimum absolute non-zero value of right hand sides passed to the solver

Available: Attribute statement (use after solve)

rObj (real): Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish

Available: Attribute statement (use after solve)

This model attribute returns the objective function value from the relaxed solve of a MIP when the integer solver did not finish.

save (string): Creates a work file, see The Save and Restart Feature

Synonym: S

Available: Command line

This option specifies the name of a work file to be written. The work file is intended to be used later to restart the GAMS program and it is also frequently referred to as save file or restart file. If no explicit file extension is provided, the default file extension .g00 is used. If no specific file path is provided the work file is created in the current directory.

Note
- The character "?" is not allowed in save file names - that use was intended for old (pre-GAMS 21.7) save files only.
- Save files are platform-independent.

For further information including examples, see chapter The Save and Restart Feature.

saveObfuscate (string): Creates an obfuscated work file, see Obfuscated Work Files

Synonym: SO

Available: Command line

savePoint (integer): Save solver point in GDX file

Synonym: SP

Available: Command line, Option statement, Attribute statement (use before solve)

This option instructs GAMS to save a point format GDX file that contains the information on the current solution point.

Default: 0
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No point GDX file is to be saved</td>
</tr>
<tr>
<td>1</td>
<td>A point GDX file from the last solve is to be saved</td>
</tr>
<tr>
<td>2</td>
<td>A point GDX file from every solve is to be saved</td>
</tr>
<tr>
<td>3</td>
<td>A point GDX file from the last solve is to be saved in the scratch directory</td>
</tr>
<tr>
<td>4</td>
<td>A point GDX file from every solve is to be saved in the scratch directory</td>
</tr>
</tbody>
</table>

**scaleOpt (boolean):** Employ user specified variable and equation scaling factors

Available: Attribute statement (use before solve)

This option determines whether GAMS will employ user-specified variable and equation scaling factors. It must be set to a nonzero value if scaling factors are to be used. For more details on scaling, see section Model Scaling - The Scale Option.

Default: 0

**scrDir (string):** Scratch directory

Synonym: SD

Available: Command line

This option specifies the name of the scratch directory. The scratch directory is used by GAMS for intermediate files that are generated during execution. The scratch directory and all its contents are usually deleted at the end of the GAMS run. By default, the scratch directory takes its value from the process directory that is specified with the option procDir. If neither the scratch directory nor the process directory are specified, the scratch directory will be set to a subdirectory of the current working directory with an internally generated name. If the scratch directory is specified, the respective directory must already exist and neither the content nor the directory itself will be deleted by GAMS at the end of the run.

Note that the option solveLink may be used to reduce or eliminate the need for intermediate files.

**scrExt (string):** Scratch file extension to be used with temporary files

Synonym: SE

Available: Command line

This option specifies the name of the extension for the temporary files that GAMS is generating during execution.

Default: dat

**scriptExit (string):** Program or script to be executed at the end of a GAMS run

Available: Command line

By default, GAMS does not call an exit script anymore. If this is required, the option scriptExit has to be set explicitly to the script that should be called after GAMS terminates. Note that an empty template of an exit script is in the GAMS system directory: it is called gmsxitnt.cmd for Windows and gmsxitus.run for Unix.

**scriptFirst (string):** First line to be written to GAMSNEXT file.
Synonym: SF

Available: Command line

This option specifies the first line written to `gamsnext`. The default is an empty string and the first line is not written.

`scrNam (string):` Work file names stem

Synonym: SN

Available: Command line

This option specifies the name stem that is used to complete the names of intermediate work files. Note that the name stem must have at least one ‘?’. The name will be completed with the scratch directory and the standard scratch name extension.

`seed (integer):` Random number seed

Available: Command line, Option statement

This option specifies the seed that is used for the pseudo random number generator.

Default: `3141`

`showOSMemory (integer):` Show the memory usage reported by the Operating System instead of the internal counting

Available: Command line

Default: `0`

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Show memory by internal accounting</td>
</tr>
<tr>
<td>1</td>
<td>Show resident set size reported by operating system</td>
</tr>
<tr>
<td>2</td>
<td>Show virtual set size reported by operating system</td>
</tr>
</tbody>
</table>

`solPrint (integer or string):` Solution report print option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the printing of the solution listing to the listing file.

N.B: the command line accepts the string and numeric value, the option statement only the string value, and the attribute statement accepts the numeric value and compile-time constant. The table below offers an overview of the different types of values and a description of the associated meaning.

<table>
<thead>
<tr>
<th>String Value</th>
<th>Numeric Value</th>
<th>Compile-Time Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>0</td>
<td><code>solprint.Off</code></td>
<td>Remove solution listings following solves</td>
</tr>
<tr>
<td>on</td>
<td>1</td>
<td><code>solprint.On</code></td>
<td>Include solution listings following solves</td>
</tr>
<tr>
<td>silent</td>
<td>2</td>
<td><code>solprint.Silent</code></td>
<td>Suppress all solution information</td>
</tr>
</tbody>
</table>
solSlack (boolean): Causes the equation output in the listing file to contain slack variable values instead of level values

Available: Option statement

If the value of this option is set to 1, the equation output in the listing file will contain slack variable values instead of level values.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>includes equation levels in the solution part of the LST file following solves</td>
</tr>
<tr>
<td>1</td>
<td>includes equation slacks in the solution part of the LST file following solves</td>
</tr>
</tbody>
</table>

solveLink (integer): Solver link option

Synonym: SL

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies what solver linking conventions are used when GAMS executes a solve statement. Note that values 3 and 6 are relevant for grid computing. Observe that there are compile-time constants that are associated with this option. When the model instance is saved to the scratch directory (values 0, 1, 2, and 3), the solution produced by the solve is also written to the scratch directory and read in by GAMS when the execution resumes.

Default: 2

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Model instance and entire GAMS state saved to scratch directory, GAMS exits (and vacates memory), and the solver script is called. After the solver terminates, GAMS restarts from the saved state and continues to executing</td>
</tr>
<tr>
<td>1</td>
<td>Model instance saved to scratch directory, the solver is called from a shell while GAMS remains open</td>
</tr>
<tr>
<td>2</td>
<td>Model instance saved to scratch directory, the solver is called with a spawn (if possible) or a shell (if spawn is not possible) while GAMS remains open - If this is not supported by the selected solver, it gets reset to 1 automatically</td>
</tr>
<tr>
<td>3</td>
<td>Model instance saved to scratch directory, the solver starts the solution and GAMS continues in a grid computing environment</td>
</tr>
<tr>
<td>4</td>
<td>Model instance saved to scratch directory, the solver starts the solution and GAMS waits for the solver to come back but uses same submission process as 3 (test mode)</td>
</tr>
<tr>
<td>5</td>
<td>The model instance is passed to the solver in-memory - If this is not supported by the selected solver, it gets reset to 2 automatically</td>
</tr>
<tr>
<td>6</td>
<td>The model instance is passed to the solver in-memory, the solver starts the solution and GAMS continues in a grid computing environment</td>
</tr>
<tr>
<td>7</td>
<td>The model instance is passed to the solver in-memory, the solver starts the solution and GAMS waits for the solver to come back but uses same submission process as 6 (test mode)</td>
</tr>
</tbody>
</table>

solveOpt (integer or string): Multiple solve management
Available: Option statement, Attribute statement (use before solve)

This option will instruct GAMS how to manage the model solution if only part of the variables and equations in the particular problem are solved.

Observe that this option is available as model attribute and option statement. The values for model attributes are numeric, while the values for option statements are text strings. In addition, there are compile-time constants that are associated with this option. The first of the two tables below offers an overview of how the different types of values are related and the second table gives the values in numerical terms and a description of the associated meaning.

<table>
<thead>
<tr>
<th>Numeric Value</th>
<th>String Value</th>
<th>Compile-Time Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>replace</td>
<td>solveOpt.Replace</td>
</tr>
<tr>
<td>1</td>
<td>merge</td>
<td>solveOpt.Merge</td>
</tr>
<tr>
<td>2</td>
<td>clear</td>
<td>solveOpt.Clear</td>
</tr>
</tbody>
</table>

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The solution information for all equations appearing in the model is completely replaced by the new model results; variables are only replaced if they appear in the final model</td>
</tr>
<tr>
<td>1</td>
<td>The solution information for all equations and variables is merged into the existing solution information</td>
</tr>
<tr>
<td>2</td>
<td>The solution information for all equations appearing in the model is completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals will be removed</td>
</tr>
</tbody>
</table>

**solver (string):** Default solver for all model types that the solver is capable to process

Available: Command line, Option statement

The command line parameter `solver=abc` initializes the default solver for the model types the solver "abc" is capable of to abc. This initialization is done before the default solvers of individual model types are set via command line parameters. Consider the following example:

```
> gams transp lp=conopt solver=soplex
```

Note that this GAMS call will first set the solver Soplex as the solver for the model types LP and RMIP, since these are the model types that Soplex can handle. Then Conopt will be reset as the default solver for LPs. Observe that the order of these parameters on the command line is irrelevant. If multiple occurrences of the option `solver` appear, the last entry will set the value of the option.

In addition, the solver for multiple model types may be set in the GAMS model source code via the following statement:

```
option solver = abc;
```

This statement sets the solver for the model types the solver abc can handle to abc. Note that in such an option statement the order of other solver setting options is significant. Consider the following example:

```
option lp=conopt, solver=soplex;
```
This statement will first set the solver for LPs to Conopt and in the next step to SOPLEX because SOPLEX is capable of handling the model type LP. In some cases it makes sense to set a solver twice. Consider the following example:

```
option solver=conopt, solver=cbc;
```

This option statement has the effect that models of the types CNS, DNLP, NLP, QCP, RMIQCP or RMINLP will be solved with Conopt and models of the types LP, RMIP or MIP will be solved with CBC.

Note that as usual, a specification of a solver through an option statement in the GAMS source takes precedence over a specification on the command line.

`solverCntr (string):` Solver control file name

- Synonym: SCNTR
- Available: Command line
- This option specifies the solver control file name. Note that the name is completed with the scratch directory and the scratch extension.

`solverDict (string):` Solver dictionary file name

- Synonym: SDICT
- Available: Command line
- This option specifies the solver dictionary file name. Note that the name completed with the scratch directory and the scratch extension.

`solverInst (string):` Solver instruction file name

- Synonym: SINST
- Available: Command line
- This option specifies the solver instruction file name. Note that the name is completed with the scratch directory and the scratch extension.

`solverMatr (string):` Solver matrix file name

- Synonym: SMATR
- Available: Command line
- This option specifies the solver matrix file name. Note that the name is completed with the scratch directory and the scratch extension.

`solverSolu (string):` Solver solution file name

- Synonym: SSOLU
- Available: Command line
- This option specifies the solver solution file name. Note that the name is completed with the scratch directory and the scratch extension.

`solverStat (string):` Solver status file name

- Synonym: SSTAT
- Available: Command line
- This option specifies the solver status file name. Note that the name is completed with the scratch directory and the scratch extension.

The model attribute indicates the solver termination condition. Observe that there are compile-time constants that are related to `solveStat`. Note that additional information to the values given in the table below is provided in section Solver Status.
4.40 The GAMS Call and Command Line Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Completion</td>
</tr>
<tr>
<td>2</td>
<td>Iteration Interrupt</td>
</tr>
<tr>
<td>3</td>
<td>Resource Interrupt</td>
</tr>
<tr>
<td>4</td>
<td>Terminated By Solver</td>
</tr>
<tr>
<td>5</td>
<td>Evaluation Interrupt</td>
</tr>
<tr>
<td>6</td>
<td>Capability Problems</td>
</tr>
<tr>
<td>7</td>
<td>Licensing Problems</td>
</tr>
<tr>
<td>8</td>
<td>User Interrupt</td>
</tr>
<tr>
<td>9</td>
<td>Setup Failure</td>
</tr>
<tr>
<td>10</td>
<td>Solver Failure</td>
</tr>
<tr>
<td>11</td>
<td>Internal Solver Failure</td>
</tr>
<tr>
<td>12</td>
<td>Solve Processing Skipped</td>
</tr>
<tr>
<td>13</td>
<td>System Failure</td>
</tr>
</tbody>
</table>

**stepSum** *(boolean)*: Summary of computing resources used by job steps

Available: Command line

This option controls the generation of a step summary of the processing times taken by GAMS during a given run.

For example, the call

```
> gams trnsport stepsum=1
```

will generate the following *step summaries* in the listing file:

```
STEP SUMMARY: 0.016 0.016 STARTUP
               0.005 0.005 COMPIILATION
               0.065 0.065 EXECUTION
               0.001 0.001 CLOSEDOWN
               0.087 0.087 TOTAL SECONDS
               0.089 0.089 ELAPSED SECONDS
               3.942 3.942 MAX HEAP SIZE (MB)
```

Note that this step summary will be printed before the model is sent to the solver, thus it may be found before the solve summary. The second step summary will be printed after solution, it will appear at the very end of the listing file:

```
STEP SUMMARY: 0.004 0.020 STARTUP
               0.000 0.005 COMPIILATION
               0.003 0.068 EXECUTION
               0.000 0.001 CLOSEDOWN
               0.007 0.094 TOTAL SECONDS
               0.239 0.328 ELAPSED SECONDS
               2.899 3.942 MAX HEAP SIZE (MB)
```

Observe that the first column reports the time for the individual section of the run, while the second column reports accumulated times including previous sections.

Default: 0
**strictSingleton** *(boolean):* Error if assignment to singleton set has multiple elements

Available: Command line, Option statement

This option affects the behavior of a membership assignment to a singleton set. If the value is set to zero, GAMS will not complain about a singleton set with more than one element, but will take only the first element. However, if the value is set to 1, a singleton set definition with more than one element will cause an error. Also, if this option is set as a command line parameter it initializes the state of the dollar control option `$on/offStrictSingleton`. So, it influences the singleton set checking at both compile- and execution time.

Default: 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No step summary</td>
</tr>
<tr>
<td>1</td>
<td>Step summary printed</td>
</tr>
</tbody>
</table>

**stringChk** *(integer):* String substitution options

Available: Command line

This option affects the result of the check for `%xxx%` symbols. Note that `%xxx%` symbols may be environment variables (also at execution time) or compile-time variables.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No substitution if symbol undefined and no error</td>
</tr>
<tr>
<td>1</td>
<td>Error if symbol undefined</td>
</tr>
<tr>
<td>2</td>
<td>Remove entire symbol reference if undefined and no error</td>
</tr>
</tbody>
</table>

**subSys** *(string):* Name of subsystem configuration file

Available: Command line

This option specifies the name of the configuration file that contains solver defaults and other information. This option should be used only by advanced users who attempt to override internal subsystem information. In case you want to add a solver to your GAMS system, please inspect the section Adding Solvers via gamsconfig.yaml.

**subSystems** *(no value):* Lists all solvers available as well as the current default and active solvers in the LST file

Available: Option statement

This option has the effect that all available subsystems will be displayed in the listing file. Note that a solver is considered a subsystem.

**suffixDLVars** *(string):* Switch default for "$on/offSuffixDLVars"
Available: Command line

For more info see $on/offSuffixDLVars.

Default: off
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Activate $offSuffixDLVars</td>
</tr>
<tr>
<td>on</td>
<td>Activate $onSuffixDLVars</td>
</tr>
</tbody>
</table>

**sumInfs** *(real)*: Sum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the sum of infeasibilities after a solve.

**suppress** *(boolean)*: Compiler listing option

Available: Command line

If set to 1, this option will suppress the echoing of the contents of the input file(s) to the listing file. Note that this option is similar in functionality to the dollar control option $offlisting.

Note

The dollar control options $on/offlisting will affect the echo print in the listing file only if suppress is set to zero. If suppress is set to 1, the input file(s) will not be echoed to the listing file and the dollar control options will not have any effect on the listing file.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard compiler listing</td>
</tr>
<tr>
<td>1</td>
<td>Suppress compiler listing</td>
</tr>
</tbody>
</table>

**symbol** *(string)*: Symbol table file

Available: Command line

This option specifies the name of a partial symbol table that may be written in conjunction with reference files.

**symPrefix** *(string)*: Prefix all symbols encountered during compilation by the specified string in work file

Available: Command line

**sys10** *(boolean)*: Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer

Available: Command line, Option statement

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable conversion</td>
</tr>
<tr>
<td>1</td>
<td>Enable conversion</td>
</tr>
</tbody>
</table>

**sys11** *(integer)*: Dynamic resorting if indices in assignment/data statements are not in natural order

Available: Command line, Option statement
Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic optimization/restructuring of data</td>
</tr>
<tr>
<td>1</td>
<td>No optimization</td>
</tr>
<tr>
<td>2</td>
<td>Always optimize/restructure</td>
</tr>
</tbody>
</table>

**sys12 (integer):** Pass model with generation errors to solver

Synonym: noSolveSkip

Available: Command line, Option statement

Default: 0

**sys15 (integer):** Automatic switching of data structures used in search records

Available: Command line, Option statement

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic switching to dense data structures</td>
</tr>
<tr>
<td>1</td>
<td>No switching</td>
</tr>
<tr>
<td>2</td>
<td>Always switch</td>
</tr>
<tr>
<td>1x</td>
<td>Print additional information in lst file</td>
</tr>
</tbody>
</table>

**sys16 (integer):** Disable search record memory (aka execute this as pre-GAMS 24.5)

Available: Command line, Option statement

Default: 0

**sys17 (integer):** Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)

Available: Command line, Option statement

Default: 0

**sys18 (integer):** Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers

Available: Command line, Option statement

By default GAMS accepts floating-point numbers with arbitrarily many digits and converts them to correctly-rounded double-precision values. This option selects a backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers (see also the offDigit dollar control option).

Default: 0
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use modern scheme for reading floating-point numbers</td>
</tr>
<tr>
<td>1</td>
<td>Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers</td>
</tr>
</tbody>
</table>

**sys19** *(integer)*: Disable permutation on Column Generation (aka execute this as pre-GAMS 36)

Available: Command line, Option statement
Default: 0

**sysDir** *(string)*: GAMS system directory where GAMS executables reside

Available: Command line

This option sets the GAMS system directory. It is useful if there are multiple systems installed on the machine or when GAMS is called from an external system like Visual Basic.

**sysIdent** *(real)*: Solver identification number

Available: Attribute statement (use after solve)

**sysIncDir** *(string)*: SysInclude directory

Synonym: SDIR

Available: Command line

This option specifies the name of the directory to be used by GAMS for sysinclude files that do not have a full path specification. An absolute or relative path may be specified. If this option is not set, it will be set to the GAMS system directory.

Note that if this option is set, the default system include directory will not be searched.

Attention

Only one directory may be set with the option sDir. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

> gams myfile sdir mydir

Note that GAMS will search for any referenced sysinclude file in the directory mydir.

**sysOut** *(boolean or string)*: Solver Status file reporting option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls whether additional solver generated output (the solver status file) is included in the listing file. Note that the contents of the solver status file are useful for debugging or to get additional information about a solver run. Normally, only those messages flagged by the solver as destined for the listing file will be listed. If the solver crashes or encounters any unexpected difficulties, the contents of the solver status file will be automatically sent to the listing file.

Note that the boolean values *(off: 0 and off: 1)* are deprecated, they are only relevant for backward compatibility. The table below gives the boolean values and the associated meaning.
### 4.40 The GAMS Call and Command Line Parameters

#### String Value | Meaning
--- | ---
off | Suppress additional solver generated output
on | Include additional solver generated output

Default: **off**

**sysVer** (*real*): Solver version

Available: Attribute statement (use after solve)

**tabIn** (*integer*): Tab spacing

Available: Command line

This option sets the tab spacing. By default, tabs are not allowed in GAMS. However, the most common setting is 8. As a result, the tabs are at columns 1, 9, 17, ... and the intermediate columns are replaced by blanks.

Default: **8**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tabs are not allowed</td>
</tr>
<tr>
<td>1</td>
<td>Tabs are replaced by blanks</td>
</tr>
<tr>
<td>n</td>
<td>Tabs are 1, n+1, 2n+1,... (default: n=8)</td>
</tr>
</tbody>
</table>

**tFormat** (*boolean*): Time format

**Synonym**: **TF**

Available: Command line

This option controls the time format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the time 7:45 PM will be written as 19:45:00 with the default **tf** value of zero and as 19.45.00 with **tf=1**.

Default: **0**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Time as hh:mm:ss</td>
</tr>
<tr>
<td>1</td>
<td>Time as hh.mm.ss</td>
</tr>
</tbody>
</table>

**threads** (*integer*): Number of threads to be used by a solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of threads (and thereby the number of processors) to be used by a solver. If the number is greater than the number of available processors, it will be reduced to the number of processors available. Note that a value of zero implies the number of threads used will be equal to the number of processors available.

Default: **1**
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use number of available processors</td>
</tr>
<tr>
<td>n</td>
<td>Use n threads</td>
</tr>
<tr>
<td>minus_n</td>
<td>Number of processors to leave free for other tasks</td>
</tr>
</tbody>
</table>

**threadsAsync** (*integer*): Limit on number of threads to be used for asynchronous solves (solveLink=6)

Available: Command line, Option statement

Default: -1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use number of available processors</td>
</tr>
<tr>
<td>n</td>
<td>Use n threads</td>
</tr>
<tr>
<td>minus_n</td>
<td>Number of processors to leave free for other tasks</td>
</tr>
</tbody>
</table>

**timer** (*integer*): Instruction timer threshold in milli seconds

Available: Command line

This option specifies an instruction timer threshold in milli seconds. That means that only details about internal GAMS instructions that took more than n milli seconds are echoed to the log.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interpreted as +inf, no details echoed</td>
</tr>
<tr>
<td>n</td>
<td>Echo all details about internal GAMS instructions that took more than n milli seconds to the log</td>
</tr>
</tbody>
</table>

**tolInfeas** (*real*): Infeasibility tolerance for an empty row of the form a.. 0*x =e= 0.0001;

Available: Attribute statement (use before solve)

This option specifies the infeasibility tolerance for an empty row of the following form:

a.. 0*x =e= 0.0001;

If the option is not set, a tolerance of 10 times the machine precision will be used. Empty rows that fail this infeasibility check will be flagged with the listing file message Equation infeasible due to rhs value.

**tolInfRep** (*real*): This attribute sets the tolerance for marking infeasible in the equation listing

Available: Attribute statement (use before solve)

This option sets the tolerance for marking an equation infeasible in the equation listing. Note that the default value is 1.0e-13.

Default: 1.0E-13
**tolProj** (real): Tolerance for setting solution values to a nearby bound when reading a solution

Available: Attribute statement (use before solve)

When a solution is returned from a solver, equation and variable level values that are very close to their lower or upper bounds are projected or moved to that bound, and a count of the projections/moves made is included in the solution report’s REPORT SUMMARY. This option controls the relative tolerance used in testing for closeness: project if \( \text{abs(level-bound)} < \text{tolproj} \times (1+\text{abs(level)}) \). For MCP models, this projection is also applied to the variable marginal values. In this case the projection count is not incremented and the test is an absolute and not a relative test: project if \( \text{abs(marginal)} < \text{tolproj} \).

Default: 0

**trace** (string): Trace file name

Available: Command line

This option specifies the trace file name and causes a trace file to be written. Note that if a previous trace file of the same name already exists, then all new data output will be appended. Therefore, users should be careful to delete all old versions of the trace file if the a file name is reused and they do wish the new data to be appended.

**traceLevel** (integer): Modelstat/Solvestat threshold used in conjunction with action=GT

Synonym: TL

Available: Command line

Default: 0

**traceOpt** (integer): Trace file format option

Available: Command line

This option specifies the format of the trace file. Note that several different types of trace files may be created, depending on what output information is desired.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver and GAMS step trace</td>
</tr>
<tr>
<td>1</td>
<td>Solver and GAMS exit trace</td>
</tr>
<tr>
<td>2</td>
<td>Solver trace only</td>
</tr>
<tr>
<td>3</td>
<td>Solver trace only in format used for GAMS performance world</td>
</tr>
<tr>
<td>5</td>
<td>Gams exit trace with all available trace fields</td>
</tr>
</tbody>
</table>

**tryInt** (real): Whether solver should make use of a partial integer-feasible solution

Available: Attribute statement (use before solve, reset by solve statement)

Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. The exact form of implementation depends on the solver and may be partly controlled by solver settings or options. See the solver manuals for details.
tryLinear (real): Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used

Available: Attribute statement (use before solve)

If this option is set to 1, empirical NLP models will be examined to determine if there are any active NLP terms. If there are none, the default LP solver will be used. The procedure also checks to see if QCP and DNLP models can be reduced to an LP; MIQCP and MINLP can be solved as a MIP; RMIQCP and RMINLP can be solved as an RMIP. Note that the default value is zero.

user1..5 (string): User string N

Synonym: U1

Available: Command line

This option permits users to enter a text for up to 5 user-defined options. The double dash parameters supersede these parameters.

warnings (integer): Number of warnings permitted before a run terminates

Available: Command line

This option specifies the maximum number of allowable warnings, before the run terminates.

Default: ∞

workDir (string): Working directory

Synonym: WDir

Available: Command line

This option sets the working directory. This option is useful when GAMS is called from an external system like Visual Basic. If it is not specified, the working directory will be set to the directory curDir.

workFactor (real): Memory Estimate multiplier for some solvers

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace to allocate for problem solution relative to the solver-computed estimate. For example, setting the value to 2 will double the memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect. Note that in cases where both options workfactor and workSpace are specified, the value for workSpace will take precedence.

Default: 1

workSpace (real): Work space for some solvers in MB

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace in Megabytes to allocate. If it is not specified by the user, the solver will estimate the size. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect, or it may be used as a memory limit.
4.40 The GAMS Call and Command Line Parameters

xSave (string): Creates a compressed work file

Synonym: XS
Available: Command line

This option specifies the name of a save file written in ASCII format in older GAMS systems (versions older than 21.7), in order for the save file to be platform dependent and may be moved to machines with different operating systems.

In GAMS systems from release 22.3 and newer this option has the effect that compressed save files are written.

xSaveObfuscate (string): Creates a compressed obfuscated work file

Synonym: XSO
Available: Command line

zeroRes (real): The results of certain operations will be set to zero if abs(result) LE ZeroRes

Available: Command line

This option specifies the threshold value for internal rounding to zero in certain operations.

Default: 0

zeroResRep (boolean): Report underflow as a warning when abs(results) LE ZeroRes and result set to zero

Available: Command line

This option causes GAMS to issue warnings whenever a rounding occurs because of the setting of the option zeroRes.

Default: 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No warning when a rounding occurs because of ZeroRes</td>
</tr>
<tr>
<td>1</td>
<td>Issue warnings whenever a rounding occurs because of ZeroRes</td>
</tr>
</tbody>
</table>

zeroToEps (string): Treat zero as eps

Available: Option statement

When this is set to on, zero values are interpreted as EPS when loading non-scalar parameters from GDX or embedded code at execution time. Also, when deriving a set from a parameter in a GDX file at execution time, set records will be created for parameter records with value 0, if zeroToEps is set to on, but not, if it is off.

See dollar control option $onEps for the compile-time equivalent.

Default: off

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Treat Zero as Zero</td>
</tr>
<tr>
<td>on</td>
<td>Treat Zero as Eps</td>
</tr>
</tbody>
</table>
4.40.7 Executing an External Program

External programs may be run during a GAMS job either using the $call, Execute or Put_utility syntax. The $call procedure is executed at the moment that is encountered during compilation. The Execute and Put_utility commands causes the external program to be run during GAMS program execution. The contrast between these statements is important in two ways.

- Influence on results that can be included in a GAMS program – Anything run with $Call can generate files that can be included in the subsequent compilation. On the other hand files generated with Execute and Put_utility cannot be included because $Include operates only at compile time (unless you use Save and Restart).

- Influence on results that can be fed into the external program – Obviously when one is running an external program there is the desire to pass it data depicting results of the GAMS execution. $Call cannot do this as the data passed have to exist at compile time and cannot use the result of any GAMS calculations and solves in the current program. Execute commands on the other hand can use any data generated during a run which arise before the Execute and Put_utility command's position in the file through passage via put files or other mechanisms.

The big difference between the $call and execute is

- $call
  - can generate results to be immediately incorporated back into GAMS
  - cannot use GAMS results generated within this run because the $Call is executed at compile time.

- execute and put_utility
  - can cause a program to be started using results generated by the GAMS program (note such results do have to have been saved in an external file using a command like put)
  - cannot generate results which can be immediately reincluded into the GAMS program because new material can only be added compile time.

4.40.7.1 Execute

This command uses the syntax

`execute[.async[NC] | .checkErrorLevel] "[=]command_to_execute"`

to execute a program specified by command_to_execute. The execution occurs during the GAMS execution phase.

- The = will call the program directly, while without the = GAMS calls a shell that executes the program. When a program is executed in a shell, mechanisms like redirection (>) and pipe (|) will work.
- The .async suffix makes GAMS go ahead without waiting.
- The .asyncNC option tells the operating system to start the run a new console rather than sharing the console of the parent process allowing use of multiple processors. This is available under Windows only.
- The .checkErrorLevel suffix checks the errorLevel implicitly, raises an execution error and aborts the execution, if that is not 0.
- Since this occurs during execution one cannot use the compile time $Include to incorporate the results of that external run into the GAMS code except through a GAMS from GAMS approach as discussed below or through save and restart use (see the Save Restart chapter).
4.40.7.2 Asynchronous Execution

The `.async` variant of `$\texttt{call}$` and `$\texttt{execute}$` start a job without waiting for the result. One can continue in the GAMS program and collect the return code of the job later. There are three ways to start a job asynchronously:

- `$\texttt{call\.async \ldots}$` (compile phase)
- `$\texttt{execute\.async }'\ldots''` (execution phase)
- `'put_utility fx 'exec.async' / '\ldots'; / put_utility fx 'shell.async' / '\ldots';` (execution phase)

After each of those the function `JobHandle` can be used to get the Process ID (pid) of the last job started. With `jobStatus(pid)` one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by `errorLevel`
- 3: process not running anymore or was never running, no return code available

With `jobTerminate(pid)` a interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

With `jobKill(pid)` a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

The model `[ASYNCEXEC]` from the GAMS Test Library demonstrates the use of this feature.

4.41 Dollar Control Options

4.41.1 Introduction

Dollar control options are used to indicate compiler directives and options. Dollar control options are not part of the GAMS language and must be entered on separate lines marked with the symbol `$` in the first column. A dollar control option line may be placed anywhere within a GAMS program and it is processed during the compilation of the program. The symbol `$` is followed by one or more options separated by spaces. Since the dollar control options are not part of the GAMS language, they do not appear on the compilation output in the listing file unless an error has been detected or the user has requested them to be shown (with the option `$\texttt{onDollar}$`). Note that dollar control option lines are not case sensitive and a continued compilation uses the previous settings.

This chapter is organized as follows. First an overview of the dollar control options will be given in section List of Dollar Control Options, where the options will be presented in groups reflecting their major functional categories. Section Detailed Description of Dollar Control Options will contain a reference list of all dollar control options in alphabetical order with detailed description for each.

We will conclude this chapter with separate sections for two important topics: Conditional Compilation, Macros in GAMS, Compressing and Decompressing Files, and Encrypting Files.
4.41.1 Syntax

In general, the syntax in GAMS for dollar control statements is as follows:

\$option_name argument_list {option_name argument_list}

The symbol \$ in the first column indicates that this is a dollar control statement. It is followed by the name of the dollar control option \option_name and the list of arguments \argument_list of the option. Depending on the particular option, the number of arguments required can vary from 0 to many. More than one dollar control option may be activated in one statement. Note that in this case the symbol \$ is not repeated. Observe that some dollar control options require that they be the first option on a line.

Note

- No blank space is permitted between the character \$ and the first option that follows.
- The effect of the dollar control option is felt immediately after the option is processed.
- Dollar control options are not part of the GAMS language they instruct the compiler to perform some task. Therefore, dollar control options are not terminated with a semicolon as real GAMS language statements.

A simple example of a list of dollar control options is shown below:

\$title Example to illustrate dollar control options
\$onsymxref onsymlist

Note that there is no blank space between the character \$ and the option that follows. The first dollar control option \$title sets the title of the pages in the listing file to the text that follows the option name. In the second line of the example above, two options are set: \$onSymXRef and \$onSymList. These options turn on the echoing of the symbol cross reference table and symbol listing in the compilation output in the listing file.

Observe that it is also permitted to place a dollar control statement in a column other than column 1. However, in this case the statement must begin with the symbols $$, like in this example

$$title Example showing that dollar control option can start in any column with an extra $ added

4.41.2 List of Dollar Control Options

The dollar control options are grouped into nine major functional categories affecting

- the input comment format
- the input data format
- the output format
- reference maps
- program control
- GDX operations
- compile-time variables and environment variables
- macro definitions
- compressing and encrypting source files

The following subsections briefly describe the options in each of the categories.

4.41.2.1 Dollar Control Options Affecting the Input Comment Format
Note that comments in GAMS are introduced in section Comments.

### 4.41 Dollar Control Options Affecting the Input Data Format

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>Set the comment character</td>
</tr>
<tr>
<td>eolCom</td>
<td>Set the end-of-line comment character(s)</td>
</tr>
<tr>
<td>inlineCom</td>
<td>Set the in-line comment characters</td>
</tr>
<tr>
<td>maxCol</td>
<td>Set the right-hand margin of the input file</td>
</tr>
<tr>
<td>minCol</td>
<td>Set left-hand margin of the input file</td>
</tr>
<tr>
<td>offEolCom</td>
<td>Turn end-of-line comments off</td>
</tr>
<tr>
<td>offInline</td>
<td>Turn in-line comments off</td>
</tr>
<tr>
<td>offMargin</td>
<td>Turn margin marking off</td>
</tr>
<tr>
<td>offNestCom</td>
<td>Turn nested comments off</td>
</tr>
<tr>
<td>offText</td>
<td>Turn text mode off</td>
</tr>
<tr>
<td>onEolCom</td>
<td>Turn end-of-line comments on</td>
</tr>
<tr>
<td>onInline</td>
<td>Turn in-line comments on</td>
</tr>
<tr>
<td>onMargin</td>
<td>Turn margin marking on</td>
</tr>
<tr>
<td>onNestCom</td>
<td>Turn nested comments on</td>
</tr>
<tr>
<td>onText</td>
<td>Turn text on: the following lines are comment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dollar</td>
<td>Set the 'dollar' character</td>
</tr>
<tr>
<td>offDelim</td>
<td>Turn delimited data statement syntax off</td>
</tr>
<tr>
<td>offDigit</td>
<td>Turn the number precision check off</td>
</tr>
<tr>
<td>offEmbedded</td>
<td>No embedded text or data allowed</td>
</tr>
<tr>
<td>offEmpty</td>
<td>Disallow empty data initialization statements</td>
</tr>
<tr>
<td>offEnd</td>
<td>Disallow alternate program control syntax</td>
</tr>
<tr>
<td>offEps</td>
<td>Disallow interpretation of $EPS$ as zero</td>
</tr>
<tr>
<td>offExternalInput</td>
<td>End of external input section</td>
</tr>
<tr>
<td>offGlobal</td>
<td>Disallow inheritance of parent file settings</td>
</tr>
<tr>
<td>offIDCProtect</td>
<td>Allow to change external input symbols at execution time</td>
</tr>
<tr>
<td>offSuffixDLVars</td>
<td>Do not allow domain limited variables with suffixes in model</td>
</tr>
<tr>
<td>offUNDF</td>
<td>Do not allow $UNDF$ as input</td>
</tr>
<tr>
<td>offUni</td>
<td>Do not allow domain violations in assignments</td>
</tr>
<tr>
<td>offWarning</td>
<td>Do not convert domain errors into warnings</td>
</tr>
<tr>
<td>onDelim</td>
<td>Turn delimited data statement syntax on</td>
</tr>
<tr>
<td>onDigit</td>
<td>Turn number precision check on</td>
</tr>
<tr>
<td>onEmbedded</td>
<td>Allow embedded text or data in set and parameter statements</td>
</tr>
<tr>
<td>onEmpty</td>
<td>Allow empty data initialization statements</td>
</tr>
<tr>
<td>onEnd</td>
<td>Allow alternate program control syntax</td>
</tr>
<tr>
<td>onEps</td>
<td>Interpret zero as $EPS$</td>
</tr>
<tr>
<td>onExternalInput</td>
<td>Start of external input section</td>
</tr>
<tr>
<td>onGlobal</td>
<td>Force inheritance of parent file settings</td>
</tr>
<tr>
<td>onIDCProtect</td>
<td>Protect external input symbols from being changed at execution time</td>
</tr>
<tr>
<td>onSuffixDLVars</td>
<td>Allow domain limited variables with suffixes in model</td>
</tr>
<tr>
<td>onUNDF</td>
<td>Allow $UNDF$ as input</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>onUni</td>
<td>Allow domain violations in assignments</td>
</tr>
<tr>
<td>onWarning</td>
<td>Convert certain domain errors into warnings</td>
</tr>
<tr>
<td>use205</td>
<td>Language syntax of release 2.05</td>
</tr>
<tr>
<td>use225</td>
<td>Language syntax of release 2.25 Version 1</td>
</tr>
<tr>
<td>use999</td>
<td>Latest language syntax</td>
</tr>
<tr>
<td>version</td>
<td>Test GAMS compiler version number</td>
</tr>
</tbody>
</table>

### 4.41.2.3 Dollar Control Options Affecting the Output Format

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>Double-spaced listing follows</td>
</tr>
<tr>
<td>echo</td>
<td>Echo text to a file</td>
</tr>
<tr>
<td>echoN</td>
<td>Echo text to a file without ending the line</td>
</tr>
<tr>
<td>eject</td>
<td>Advance to next page</td>
</tr>
<tr>
<td>hidden</td>
<td>Ignore text and do not list</td>
</tr>
<tr>
<td>lines</td>
<td>Next number of lines have to fit on the page</td>
</tr>
<tr>
<td>log</td>
<td>Send message to the log</td>
</tr>
<tr>
<td>offDollar</td>
<td>Turn the listing of dollar control option lines off</td>
</tr>
<tr>
<td>offEcho</td>
<td>End of block echo</td>
</tr>
<tr>
<td>offExternalOutput</td>
<td>End of external output section</td>
</tr>
<tr>
<td>offEpsToZero</td>
<td>Do not interpret EPS as zero when writing to GDX</td>
</tr>
<tr>
<td>offInclude</td>
<td>Turn the listing of include file names off</td>
</tr>
<tr>
<td>offListing</td>
<td>Turn echoing input lines to listing file</td>
</tr>
<tr>
<td>offLog</td>
<td>Turn line logging off</td>
</tr>
<tr>
<td>offPut</td>
<td>End of block put</td>
</tr>
<tr>
<td>offUpper</td>
<td>Following print to listing file is mixed cased</td>
</tr>
<tr>
<td>offVerbatim</td>
<td>Stop verbatim copy</td>
</tr>
<tr>
<td>onDollar</td>
<td>Turn the listing of dollar control option lines on</td>
</tr>
<tr>
<td>onEcho</td>
<td>Start of block echo with substitution</td>
</tr>
<tr>
<td>onEchoS</td>
<td>Start of block echo with substitution</td>
</tr>
<tr>
<td>onEchoV</td>
<td>Start of block echo without substitution</td>
</tr>
<tr>
<td>onEpsToZero</td>
<td>Interpret EPS as zero when writing to GDX</td>
</tr>
<tr>
<td>onExternalOutput</td>
<td>Start of external output section</td>
</tr>
<tr>
<td>onInclude</td>
<td>Include file name echoed to listing file</td>
</tr>
<tr>
<td>onListing</td>
<td>Input lines echoed to listing file</td>
</tr>
<tr>
<td>onLog</td>
<td>Reset line logging</td>
</tr>
<tr>
<td>onPut</td>
<td>Start of block put without substitution</td>
</tr>
<tr>
<td>onPutS</td>
<td>Start of block put with substitution</td>
</tr>
<tr>
<td>onPutV</td>
<td>Start of block put without substitution</td>
</tr>
<tr>
<td>onUpper</td>
<td>Following print to listing file is all upper cased</td>
</tr>
<tr>
<td>onVerbatim</td>
<td>Start verbatim copy if dumpopt $\geq 10$</td>
</tr>
<tr>
<td>remark</td>
<td>Comment line with suppressed line number</td>
</tr>
<tr>
<td>single</td>
<td>Single-spaced listing follows</td>
</tr>
<tr>
<td>stars</td>
<td>Set &quot;****&quot; characters in listing file</td>
</tr>
<tr>
<td>sTitle</td>
<td>Set subtitle and reset page</td>
</tr>
<tr>
<td>title</td>
<td>Set title and reset page</td>
</tr>
</tbody>
</table>
4.41.2.4 Dollar Control Options Affecting the Listing of Reference Maps

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offSymList</td>
<td>Turn symbol list off</td>
</tr>
<tr>
<td>offSymXRef</td>
<td>Turn symbol cross reference listing off</td>
</tr>
<tr>
<td>offUElList</td>
<td>Turn unique element listing off</td>
</tr>
<tr>
<td>offUElXRef</td>
<td>Turn unique element cross reference off</td>
</tr>
<tr>
<td>onSymList</td>
<td>Turn symbol list on</td>
</tr>
<tr>
<td>onSymXRef</td>
<td>Turn symbol cross reference listing on</td>
</tr>
<tr>
<td>onUElList</td>
<td>Turn unique element listing on</td>
</tr>
<tr>
<td>onUElXRef</td>
<td>Turn unique element cross reference on</td>
</tr>
</tbody>
</table>

4.41.2.5 Dollar Control Options Affecting Program Control

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort[.noError]</td>
<td>Issue an (error) message and abort compilation</td>
</tr>
<tr>
<td>batInclude</td>
<td>Include file with substitution arguments</td>
</tr>
<tr>
<td>call</td>
<td>Execute another program during compilation</td>
</tr>
<tr>
<td>call.Async[NC]</td>
<td>Execute another program asynchronously during compilation</td>
</tr>
<tr>
<td>call.checkErrorLevel</td>
<td>Execute another program during compilation and raise error in case of problem</td>
</tr>
<tr>
<td>clear</td>
<td>Reset all data for an identifier to its default value</td>
</tr>
<tr>
<td>clearError[s]</td>
<td>Clear compilation errors</td>
</tr>
<tr>
<td>else</td>
<td>Else clause</td>
</tr>
<tr>
<td>elseif</td>
<td>ElseIf structure with case sensitive comparison</td>
</tr>
<tr>
<td>elseifE</td>
<td>ElseIf structure with expression evaluation</td>
</tr>
<tr>
<td>elseifI</td>
<td>ElseIf structure with case insensitive comparison</td>
</tr>
<tr>
<td>endif</td>
<td>Close ifThen/ifThenE/ifThenI control structure</td>
</tr>
<tr>
<td>error</td>
<td>Issue an error message</td>
</tr>
<tr>
<td>exit</td>
<td>Exit from compilation</td>
</tr>
<tr>
<td>funcLibIn</td>
<td>Load extrinsic function library</td>
</tr>
<tr>
<td>goto</td>
<td>Go to line with given label name</td>
</tr>
<tr>
<td>hiddenCall</td>
<td>Execute another program (hidden) during compilation</td>
</tr>
<tr>
<td>hiddenCall.Async[NC]</td>
<td>Execute another program (hidden) asynchronously during compilation</td>
</tr>
<tr>
<td>hiddenCall.checkErrorLevel</td>
<td>Execute another program (hidden) during compilation and raise error in case of problem</td>
</tr>
<tr>
<td>if</td>
<td>Conditional processing, case sensitive</td>
</tr>
<tr>
<td>ifE</td>
<td>If statement with expression evaluation</td>
</tr>
<tr>
<td>ifI</td>
<td>Conditional processing, case insensitive</td>
</tr>
<tr>
<td>ifThen</td>
<td>IfThen-elseif structure with case sensitive comparison</td>
</tr>
<tr>
<td>ifThenE</td>
<td>IfThen-elseif structure with expression evaluation</td>
</tr>
<tr>
<td>ifThenI</td>
<td>IfThen-elseif structure with case insensitive comparison</td>
</tr>
<tr>
<td>include</td>
<td>Include file</td>
</tr>
<tr>
<td>kill</td>
<td>Kill data connected with identifier</td>
</tr>
<tr>
<td>label</td>
<td>Label name as entry point from $goto$</td>
</tr>
<tr>
<td>libInclude</td>
<td>Include file from library directory</td>
</tr>
<tr>
<td>offCheckErrorLevel</td>
<td>Do not check $errorLevel$ automatically after $[hidden]call</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offECImplicitLoad</td>
<td>Do not allow implicit loading from embedded code</td>
</tr>
<tr>
<td>offEmbeddedCode</td>
<td>Ends embedded code section</td>
</tr>
<tr>
<td>offFiltered</td>
<td>Turn behavior of <code>$gdx[load]</code> to domain checked read</td>
</tr>
<tr>
<td>offMulti</td>
<td>Turn redefinition of data off</td>
</tr>
<tr>
<td>offOrder</td>
<td>Allow <code>lag</code> and <code>lead</code> operations on dynamic or unordered sets</td>
</tr>
<tr>
<td>offRecurse</td>
<td>Disable recursive include files</td>
</tr>
<tr>
<td>offStrictSingleton</td>
<td>Take first label if data statement for singleton set has multiple elements</td>
</tr>
<tr>
<td>onCheckErrorLevel</td>
<td>Throw compilation error, if <code>errorLevel</code> is not 0 after <code>$[hidden]call</code></td>
</tr>
<tr>
<td>onECImplicitLoad</td>
<td>Allow implicit loading from embedded code</td>
</tr>
<tr>
<td>onEmbeddedCode</td>
<td>Starts embedded code section with substitution</td>
</tr>
<tr>
<td>onEmbeddedCodeS</td>
<td>Starts embedded code section with substitution</td>
</tr>
<tr>
<td>onFilterd</td>
<td>Turn behavior of <code>$[gdx][load]</code> to filtered read</td>
</tr>
<tr>
<td>onMulti</td>
<td>Turn redefinition of data on (merging into existing data)</td>
</tr>
<tr>
<td>onMultiR</td>
<td>Turn redefinition of data on (replacing existing data)</td>
</tr>
<tr>
<td>onOrder</td>
<td><code>lag</code> and <code>lead</code> operations on constant and ordered sets only</td>
</tr>
<tr>
<td>onRecurse</td>
<td>Enable recursive include files</td>
</tr>
<tr>
<td>onStrictSingleton</td>
<td>Error if data statement for singleton set has multiple elements</td>
</tr>
<tr>
<td>maxGoTo</td>
<td>Maximum number of jumps to the same label</td>
</tr>
<tr>
<td>phantom</td>
<td>Define a phantom element</td>
</tr>
<tr>
<td>shift</td>
<td><code>bat/lib/sysInclude</code> argument shift operation</td>
</tr>
<tr>
<td>stop</td>
<td>Stop compilation</td>
</tr>
<tr>
<td>sysInclude</td>
<td>Include file from system directory</td>
</tr>
<tr>
<td>terminate</td>
<td>Terminate compilation and execution</td>
</tr>
<tr>
<td>warning</td>
<td>Issue compilation warning</td>
</tr>
</tbody>
</table>

Note that conditional compilation in GAMS is discussed in section Conditional Compilation below.

### 4.41.2.6 Dollar Control Options for GDX Operations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdxIn</td>
<td>Open GDX file for input</td>
</tr>
<tr>
<td>gdxLoad</td>
<td>Load symbols from specified GDX file</td>
</tr>
<tr>
<td>gdxOut</td>
<td>Open GDX file for output</td>
</tr>
<tr>
<td>gdxUnload</td>
<td>Unload symbols into specified GDX file.</td>
</tr>
<tr>
<td>load</td>
<td>Load symbols from previously opened GDX file</td>
</tr>
<tr>
<td>loadDC</td>
<td>Load symbols from previously opened GDX file - domain checked</td>
</tr>
<tr>
<td>loadDCM</td>
<td>Load symbols from previously opened GDX file - domain checked - merge</td>
</tr>
<tr>
<td>loadDCR</td>
<td>Load symbols from previously opened GDX file - domain checked - replace</td>
</tr>
<tr>
<td>loadFiltered</td>
<td>Load symbols from previously opened GDX file - domain filtered</td>
</tr>
<tr>
<td>loadFilteredM</td>
<td>Load symbols from previously opened GDX file - domain filtered - merge</td>
</tr>
<tr>
<td>loadFilteredR</td>
<td>Load symbols from previously opened GDX file - domain filtered - replace</td>
</tr>
<tr>
<td>loadM</td>
<td>Load symbols from previously opened GDX file - merge</td>
</tr>
<tr>
<td>loadR</td>
<td>Load symbols from previously opened GDX file - replace</td>
</tr>
<tr>
<td>offExternalInput</td>
<td>End of external input section</td>
</tr>
<tr>
<td>offExternalOutput</td>
<td>End of external output section</td>
</tr>
</tbody>
</table>
### 4.41 Dollar Control Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offFiltered</td>
<td>Turn behavior of ${gdx}load^*$ to domain checked read</td>
</tr>
<tr>
<td>onExternalInput</td>
<td>Start of external input section</td>
</tr>
<tr>
<td>onExternalOutput</td>
<td>Start of external output section</td>
</tr>
<tr>
<td>onFiltered</td>
<td>Turn behavior of ${gdx}load^*$ to filtered read</td>
</tr>
<tr>
<td>unLoad</td>
<td>Unload symbols into previously opened GDX file.</td>
</tr>
</tbody>
</table>

Note that GDX facilities and utilities are introduced in chapter GAMS Data eXchange (GDX).

### 4.41.2.7 Dollar Control Options for Compile-Time Variables and Environment Variables

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop</td>
<td>Drop a scoped compile-time variable</td>
</tr>
<tr>
<td>dropGlobal</td>
<td>Drop a global compile-time variable</td>
</tr>
<tr>
<td>dropLocal</td>
<td>Drop a local compile-time variable</td>
</tr>
<tr>
<td>dropEnv</td>
<td>Drop an OS environment variable</td>
</tr>
<tr>
<td>escape</td>
<td>Define the % escape symbol</td>
</tr>
<tr>
<td>eval</td>
<td>Evaluate and define a scoped compile-time variable</td>
</tr>
<tr>
<td>eval.Set</td>
<td>Evaluate and define a scoped compile-time variable based on a GAMS set</td>
</tr>
<tr>
<td>evalGlobal</td>
<td>Evaluate and define a global compile-time variable</td>
</tr>
<tr>
<td>evalGlobal.Set</td>
<td>Evaluate and define a global compile-time variable based on a GAMS set</td>
</tr>
<tr>
<td>evalLocal</td>
<td>Evaluate and define a local compile-time variable</td>
</tr>
<tr>
<td>evalLocal.Set</td>
<td>Evaluate and define a local compile-time variable based on a GAMS set</td>
</tr>
<tr>
<td>prefixPath</td>
<td>Prefix the environment variable PATH.</td>
</tr>
<tr>
<td>scratchFileName</td>
<td>Set a name for a temporary file in the scratch directory using the scratch file extension</td>
</tr>
<tr>
<td>setArgs</td>
<td>Define local compile-time variables using argument list</td>
</tr>
<tr>
<td>setComps</td>
<td>Unpack dotted names into compile-time variables</td>
</tr>
<tr>
<td>setDDList</td>
<td>Check double dash GAMS parameters</td>
</tr>
<tr>
<td>setDDList.Cont</td>
<td>Start list of identifiers to check double dash GAMS parameters</td>
</tr>
<tr>
<td>setEnv</td>
<td>Define an OS environment variable</td>
</tr>
<tr>
<td>set</td>
<td>Define a scoped compile-time variable</td>
</tr>
<tr>
<td>setGlobal</td>
<td>Define a global compile-time variable</td>
</tr>
<tr>
<td>setLocal</td>
<td>Define a local compile-time variable</td>
</tr>
<tr>
<td>setNames</td>
<td>Unpack a filename into local compile-time variables</td>
</tr>
<tr>
<td>show</td>
<td>Show current GAMS compile-time variables, macros, and active files</td>
</tr>
<tr>
<td>showFiles</td>
<td>Show active input and include files</td>
</tr>
<tr>
<td>showMacros</td>
<td>Show current macros</td>
</tr>
<tr>
<td>showVariables</td>
<td>Show current GAMS compile-time variables</td>
</tr>
<tr>
<td>splitOption</td>
<td>Unpack a key/value pair into scoped environment variables</td>
</tr>
</tbody>
</table>

See also sections Compile-Time Variables and Environment Variables in GAMS.

### 4.41.2.8 Dollar Control Options for Macro Definitions
Note that macros are introduced in section *Macros in GAMS* below.

### 4.41.2.9 Dollar Control Options for Compressing and Encrypting Source Files

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>Create compressed GAMS system file</td>
</tr>
<tr>
<td>decompress</td>
<td>Decompress a GAMS system file</td>
</tr>
<tr>
<td>encrypt</td>
<td>Create an encrypted GAMS system file</td>
</tr>
<tr>
<td>expose</td>
<td>Remove all access control restrictions</td>
</tr>
<tr>
<td>hide</td>
<td>Hide objects from the user</td>
</tr>
<tr>
<td>protect</td>
<td>Protect objects from being modified by the user</td>
</tr>
<tr>
<td>purge</td>
<td>Remove the objects and all associated data</td>
</tr>
</tbody>
</table>

### 4.41.3 Detailed Description of Dollar Control Options

In this section we will describe each dollar control option in detail. Note that the dollar control options are listed in alphabetical order for easy reference. Note further, that in each entry the default value, if applicable, is given in parentheses.

#### $\text{abort}[, \text{noError}]$

**Syntax:**

\[
\text{\$abort[, noError]} [\text{text}]
\]

If used as `$\text{abort}$`, this option will issue a compilation error and abort the compilation. It may be followed by a text.

**Example:**

\[
\text{\$if not \%system.fileSys\% == UNIX } \\
\text{\$abort We only do UNIX}
\]

This stops compilation if the operating system is not Unix. Running the example above on Windows will result in the compilation being aborted and the following lines in the listing file:
2. \$abort We only do UNIX

Error Messages
343 Abort triggered by above statement

This option has a variant: \$abort.noError. If the extension .noError is used in the compilation, the compilation will be aborted as well, but there will be no error. If a save file is written, all remaining unexecuted code will be flushed. This allows effective reuse of the save file.

Note that there is also an abort statement in GAMS, it is used to terminate the execution of a program.

See also \$exit, \$error, \$stop, and \$terminate.

\$batInclude

Syntax:

\$batInclude external_file {arg}

The \$batInclude facility performs the same task as the \$include facility: it inserts the contents of the specified file external_file at the location of the call. However, in addition, the option \$batInclude also passes on arguments arg which may be used inside the include file. External_file is the name of the batch include file, it may be quoted or unquoted. The arguments arg are passed on to the batch include file. These arguments are treated as character strings that are substituted by numbers inside the included file. The arguments may be single unbroken strings (quoted or unquoted) or quoted multi-part strings.

Note that the syntax has been modeled after the DOS batch facility. Inside the batch file, a parameter substitution is indicated by using the character % followed immediately by an integer value corresponding to the order of parameters on the list where %1 refers to the first argument, %2 to the second argument, and so on. If an integer value is specified that does not correspond to a passed parameter, then the parameter flag is substituted with a null string. The parameter flag %0 is a special case that will substitute a fully expanded file name specification of the current batch included file. The flag %$ is the current $ symbol (see \$dollar). Observe that parameters are substituted independent of context and the entire line is processed before it is passed to the compiler. There is one exception: parameter flags that appear in comments are not substituted.

Attention

- GAMS requires that processing the substitutions must result in a line of less than or equal to the maximum input line length.
- The case of the passed parameters is preserved, thus it may be used in string comparisons.

Example:

\$batInclude "file1.inc" abcd "bbbb" "cccc dddd"

Note that file1.inc is included with abcd as the first parameter, bbbb as the second parameter and cccc dddd as the third parameter.

Parameter a,b,c ;
a = 1 ; b = 0 ; c = 2 ;
\$batInclude inc2.inc b a
display b ;
\$batInclude inc2.inc b c
display b ;
\$batInclude inc2.inc b "a+5"
display b ;
The external file inc2.inc contains the following line:

%1 = sqr(%2) - %2 ;

The echo print in the corresponding listing file follows:

1 Parameter a,b,c ;
2 a = 1 ; b = 0 ; c = 2 ;
BATINCLUDE C:\tmp\inc2.inc
4 b = sqr(a) - a ;
5 display b ;
BATINCLUDE C:\tmp\inc2.inc
7 b = sqr(c) - c ;
8 display b ;
BATINCLUDE C:\tmp\inc2.inc
10 b = sqr(a+5) - a+5 ;
11 display b ;

Note that the option $batInclude appears three times with different arguments. GAMS interprets the contents of the batch include file in turn as:

\[
\begin{align*}
    b &= \text{sqr}(a) - a \\
    b &= \text{sqr}(c) - c \\
    b &= \text{sqr}(a+5) - (a+5) \\
\end{align*}
\]

Note that the third call is not interpreted as \( \text{sqr}(a+5)-(a+5) \), but instead as \( \text{sqr}(a+5)-a+5 \).

The results of the display statement are shown at the end of the listing file are given below:

<table>
<thead>
<tr>
<th></th>
<th>5 Parameter b</th>
<th>=</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 Parameter b</td>
<td>=</td>
<td>2.000</td>
</tr>
<tr>
<td></td>
<td>11 Parameter b</td>
<td>=</td>
<td>40.000</td>
</tr>
</tbody>
</table>

Observe that the third call leads to \( b = \text{sqr}(6)-1+5 \), thus the final value of \( b \) is 40. Suppose the statement in the batch include file is modified to read as follows:

%1 = sqr(%2) - (%2) ;

With this modification the output generated by the display statement will be as follows:

<table>
<thead>
<tr>
<th></th>
<th>5 Parameter b</th>
<th>=</th>
<th>0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 Parameter b</td>
<td>=</td>
<td>2.000</td>
</tr>
<tr>
<td></td>
<td>11 Parameter b</td>
<td>=</td>
<td>30.000</td>
</tr>
</tbody>
</table>

Note that the third call leads to \( b = \text{sqr}(6)-6 \) which results in \( b \) taking a value of 30.

Note

The option $batInclude without any arguments is equivalent to the option $include.

See also $include, $include, $sysInclude.

$call

Syntax:

$call [=]command
This option passes a command to the current operating system command processor and interrupts compilation until the command has been completed. If the command string is empty or omitted, a new interactive command processor will be loaded.

Example:

$call dir

This command creates a directory listing on a PC.

Note that the command string may be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign. Compilation errors will be issued if the command or the command processor cannot be loaded and executed properly.

$call gams trnsport
$call =gams trnsport

The first call will run the model [TRNSPORT] in a new command shell. The DOS command shell does not send any return codes from the run back to GAMS. Therefore any errors in the run are not reported back. The second call, however, will send the command directly to the system. The return codes from the system will be intercepted correctly and they will be available to the GAMS system through the errorLevel function.

Attention

• Some commands (like copy on a PC and cd in Unix) are shell commands and cannot be spawned off to the system. Using these in a system call will create a compilation error.

  $call 'copy myfile.txt mycopy.txt'
  $call '=copy myfile.txt mycopy.txt'

  The first call will work on a PC, but the second will not. The copy command may be used only from a command line shell. The system is not aware of this command (Try this command after clicking Run under the Start menu in Windows. You will find that it does not work).

• One has to take special care when the first argument of the command contains spaces (for example an executable with absolute path), e.g.:

  $call C:\path with spaces\gams.exe trnsport.gms

  This would fail, since the command processor would try to call C:\path as the executable. So, one might think that protecting the spaces using quotes will resolve this problem:

  $call "C:\path with spaces\gams.exe" trnsport.gms

  However, the quotes delimit the entire $call command, so that trnsport.gms gets interpreted as new dollar control option instead of an argument for the executable. The solution is to use nested quotes: One pair to delimit the $call command and another one to protect the spaces of the executable like this:

  $call ""C:\path with spaces\gams.exe" trnsport.gms"

  It gets even more complicated if the argument contains spaces as well, e.g. if we have my model.gms instead of trnsport.gms. Another pair of quotes solves this:

  $call """C:\path with spaces\gams.exe" my model.gms"

See also $call.Async, $call.checkErrorLevel, $hiddenCall.

$call.Async[NC]

Syntax:

$call.Async[NC] command
$\text{call.Async}$ works like $\text{call}$ but allows asynchronous job handling. This means users may start a job command without having to wait for the result, they may continue in their model and collect the return code of the job later. The function jobHandle may be used to get the process ID (pid) of the last job started. The status of the job may be checked using the function jobStatus(pid). An interrupt signal to a running job may be sent with the function jobTerminate(pid). With the function jobKill(pid) a kill signal may be sent to a running job.

The difference between $\text{call.Async}$ and $\text{call.AsyncNC}$ is, that the latter starts processes in a new console, rather than sharing the console of the parent process.

Note

On non-Windows platforms $\text{call.AsyncNC}$ and $\text{call.Async}$ are synonyms.

$\text{call.checkErrorLevel}$

Syntax:

$\text{call.checkErrorLevel [=}command$\n
$\text{call.checkErrorLevel}$ works like $\text{call}$ but checks the errorLevel implicitly, raises a compilation error and aborts compilation, if that is not 0.

Example:

$\text{call.checkErrorLevel gams trnsport}$

This is doing the same as:

$\text{call gams trnsport}$

$\text{ifE errorLevel<>0 abort 'Problem calling gams trnsport'}$

$\text{clear}$

Syntax:

$\text{clear ident {ident}}$

This option resets all data for the identifiers ident to their default values. Note that only the following data types may be reset: sets, parameters, variables and equations. Note further, that the clearing is carried out during compile time and not when the GAMS program executes.

Example:

Set i / 1*20 /;
Scalar a / 2 /;
$\text{clear i a}$
display i, a;

The option $\text{clear}$ resets i and a to their default values: an empty set for i and zero for a. The output generated by the display statement follows:

---- 4 SET i         ( EMPTY )

---- 4 PARAMETER a = 0.000
Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with `$clear` since data can be both initialized and assigned.

Scalar a / 12 /;
a = 5;
$c\text{lear} a
display a;

Note that the scalar data initialization statement is processed during compilation and the assignment statement $a = 5;$ during execution. In the order that it is processed, the example above is read by GAMS as:

* compilation step
Scalar a /12/ ;
$c\text{lear} a
* execution step
a = 5;
display a ;

Therefore the result is that $a$ takes the value of 5. The output from the display statement is as follows:

    4 PARAMETER a = 5.000

Compare also `$kill` and the execution time option `clear`.

$\text{clearError}[s]$

Syntax:

$\text{clearError}[s]$

This option ($\text{clearError}$ and $\text{clearErrors}$ are synonyms) clears GAMS awareness of compiler errors and turn them into warning messages instead.

Example:

Scalar z / 11 /;
$\text{eval} x \sqrt{(-1)}$
$c\text{learError}$
$log \%x\%$
Display z;

Note that without the use of $\text{clearError}$ the program above would not continue with the execution after line 2.

$\text{comment } (^*)$

Syntax:

$\text{comment } \text{char}$

This option changes the symbol indicating a single line comment from the default $*$ to the single character $\text{char}$. Note that after this option is used, the new comment character $\text{char}$ cannot be used in column 1 as before, since it got a special meaning. Note further, that the case of the character does not matter if it is used as a comment character. This option should be used with great care and we recommend to reset the symbol quickly to the default.
Attention

The case of the start-of-line comment character does not matter when being used.

Example:

```gams
$comment c
  c now we use a FORTRAN style comment symbol
$comment *
  * now we are back to the default
```

See also section Comments.

$compress

Syntax:

```
$compress source target
```

This option causes the file `source` to be compressed into the packed file `target`.

Example: Consider the following example where the well-known model [TRNSPORT] is used:

```
$call gamslib trnsport
$compress trnsport.gms t2.gms
$include t2.gms
```

The first command retrieves the file `trnsport.gms` and the second command compresses it. Note that a compressed GAMS file is treated like any other GAMS file, therefore it may be included and executed as usual. Large data files that do not change often can be compressed this way to save disk space.

The following example serves as a little utility to compress and decompress files:

```
$ifthen set decompress
  "$ if not set input $set input file_c.gms
  "$ if not exist %input% $abort No file input file %input% exist
  "$ if not set output $set output file.gms
  "$ log Decompressing %input% into %output%
  "$ decompress %input% %output%
$else
  "$ if not set input $set input file.gms
  "$ if not exist %input% $abort No file input file %input% exist
  "$ if not set output $set output file_c.gms
  "$ log Compressing %input% into %output%
  "$ compress %input% %output%
$endif
```

The program (saved to a file called `compress.gms`) can be used as follows:

```
> gams compress.gms --input myfile.gms --output myfile_c.gms
> gams compress.gms --decompress=1 --input myfile_c.gms --output myfile.gms
```

See also `$decompress`. Further details are given in chapter Compressing and Decompressing Files.

$decompress
4.41 Dollar Control Options

Syntax:

$decompress source target

This option causes the compressed file source to be decompressed into the unpacked file target.

Example: Consider the following example where the well-known model [TRNSPORT] is used:

$call gamslib trnsport
$compress trnsport.gms t2.gms
$decompress t2.gms t3.gms
$call diff t3.gms trnsport.gms
$if errorlevel 1 $abort t3.gms and trnsport.gms are not identical!

The first command retrieves the file trnsport.gms, the second command compresses it and the third command decompresses the compressed file. Note that the resulting file, t3.gms, is identical to the original file trnsport.gms which is tested via the diff program.

See also $compress. Further details are given in chapter Compressing and Decompressing Files.

$dollar ($)

Syntax:

$dollar char

This option changes the current 'dollar' symbol to the single character char.

Note

The special %$ substitution symbol can be used to get the current 'dollar' symbol.

Example:

$dollar #
#log now we can use '%$' as the '$' symbol

$double

Syntax:

$double

The lines following this option will be echoed double spaced to the echo print in the listing file.

Example:

Set  i / 1*2 / ;
Scalar a / 1 / ;
$double
Set  j / 10*15 / ;
Scalar b / 2 / ;

The resulting echo print in the listing file looks as follows:
1 Set i /1*2/ ;
2 Scalar a /1/ ;
4 Set j /10*15/ ;
5 Scalar b /2/ ;

Note that lines before the option $double are listed single spaced, while the lines after the option are listed with double space.

See also $single.

$drop

Syntax:

$drop VARNAME

This option destroys (removes from the program) the scoped compile-time variable VARNAME that was defined with the dollar control option $set.

Example:

$set NAME my name
$if set NAME $log Scoped compile-time variable NAME is set to "%NAME%"
$drop NAME
$if not set NAME $log Scoped compile-time variable NAME is not available anymore

See also $set, $dropGlobal, and $dropLocal.

$dropEnv

Syntax:

$dropEnv VARNAME

This dollar control option destroys (removes from the program) the operating system environment variable VARNAME. For detailed information, see the dollar control option.

Example:

$if setEnv GDXCOMPRESS $dropEnv GDXCOMPRESS

See also $setEnv, and $if setEnv.

$dropGlobal

Syntax:

$dropGlobal VARNAME

This option destroys (removes from the program) the global compile-time variable VARNAME that was defined with the dollar control option $setGlobal.

Example:
4.41 Dollar Control Options

=setGlobal NAME my name
$if setGlobal NAME $log Global compile-time variable NAME is set to "%NAME%"
=dropGlobal NAME
$if not setGlobal NAME $log Global compile-time variable NAME is not available anymore

See also $setGlobal, and $drop.

$dropLocal

Syntax:
$dropLocal VARNAME

This option destroys (removes from the program) the local compile-time variable VARNAME that was defined with the dollar control option $setLocal.

=setLocal NAME my name
$if setLocal NAME $log Local compile-time variable NAME is set to "%NAME%"
=dropLocal NAME
$if not setLocal NAME $log Local compile-time variable NAME is not available anymore

See also $setLocal, and $drop.

$echo

Syntax:
$echo text >[>] external_file

This option allows to write the text text to a file external_file. The text and the file name may both be quoted or unquoted. The file name is expanded using the working directory. The option $echo tries to minimize file operations by keeping the file open in anticipation of another $echo to be appended to the same file. The file will be closed at the end of the compilation or when an option $call or any variant of the option $include is encountered. The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file respectively.

Example:

$echo > echo.txt
$echo 'The message written goes from the first non blank' >> echo.txt
$echo 'to the first > or >> symbol unless the text is' >> echo.txt
$echo "is quoted. The input File is %gams.input%. The" >> echo.txt
$echo 'file name "echo.txt" will be completed with' >> echo.txt
$echo %gams.workdir%. >> echo.txt
$echo >> echo.txt

The content of the resulting file echo.txt is the following:

The message written goes from the first non blank to the first > or >> symbol unless the text is is quoted. The input File is C:\tmp\echoTest.gms. The file name "echo.txt" will be completed with C:\tmp\. 

See also $on/offEcho, and $echoN.

$echoN
Syntax:

$echoN text >[>] external_file

This option sends a text message text to an file external_file like $echo but writes no end of line marker so the line is repeatedly appended to by subsequent commands. The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file respectively. Note that the text and the file name may be quoted or unquoted. By default the file will be saved in the working directory.

Example:

$echoN 'Text to be sent' > 'aaa.txt'
$echoN 'More text' >> aaa.txt
$echoN And more and more and more >> aaa.txt
$echo This was entered with $echo >> 'aaa.txt'
$echo This too >> aaa.txt

The created file aaa.txt contains the following text:

Text to be sent
More text
And more and more and more
This was entered with $echo
This too

See also $on/offEcho, and $echo.

$eject

Syntax:

$eject

This option advances the echo print to the next page.

Example:

$eject
Set i,j ;
Parameter Data(i,j) ;
$eject
Scalar a;
a = 7;

The statements following the first $eject will be listed on one page in the echo print of the listing file and the statements following the second $eject will be listed on the next page.

$else

Syntax:

$ifThen[ElI] cond
...
{ $elseif[ElI] cond
  ...
} [$else
  ...
]$endif
This option always appears together with the option $IfThen[E/I]$. It is followed by an instruction which is executed if the conditional expression of the matching option $IfThen[E/I]$ is not true. For an example, see section Conditional Compilation with $IfThen and $else.

See also $IfThen, $elseIf and section Conditional Compilation.

$elseIf

Syntax:

$ifThen[E|I] cond
...
{$elseIf[E|I] cond
...
[ $else
...
$]}
$endIf

This option always appears together with the option $IfThen[E/I]. It is followed by another condition and instruction. For an example, see section Conditional Compilation with $IfThen and $else.

See also $IfThen, $else, $elseIfE, $elseIfI and section Conditional Compilation.

$elseIfE

Syntax:

$ifThen[E|I] cond
...
{$elseIf[E|I] cond
...
[ $else
...
$]}
$endIf

This option does the same as $elseIf but evaluates numerical values of the control variables.

See also $elseIf and section Conditional Compilation.

$elseIfI

Syntax:

$ifThen[E|I] cond
...
{$elseIf[E|I] cond
...
[ $else
...
$]}
$endIf

This option does the same as $elseIf but it is case insensitive.

See also $elseIf and section Conditional Compilation.

$encrypt

Syntax:
$encrypt source target

This option causes a file to be converted into an encrypted file. Here source is the name of the source file to be encrypted and target is the name for the resulting encrypted file. Note that encryption requires the secure option to be licensed and is available for commercial licenses only. The command line parameter pLicense specifies the target license to be used for encryption. The encrypted file can only run on a system licensed with the license file used for encryption. No special action is required on the executing system since GAMS recognizes whether a file is encrypted and will process it accordingly. There is no option to decrypt an encrypted file, so better keep the original unencrypted file.

Further details and examples are given in chapter Encrypting Files.

$endif

Syntax:

$ifThen[E|I] cond
...
{ $elseif[E|I] cond
  ...
}
[ $else
  ...
]$endif

This must option must be matched with one of the options $ifThen, $ifThenE or $ifThenI. For an example, see section Conditional Compilation with $ifThen and $else.

See also $ifThen and section Conditional Compilation.

$eolCom (!!!)

Syntax:

$eolCom char[char]

This option redefines and activates the end-of-line comment symbol, which may be one character or a sequence of two characters. By default, this is initialized to !!, but is not active. The option $onEolCom is used to activate the end-of-line comments. If $eolCom is used, $onEolCom is set automatically.

Example:

$eolCom //
Set i /1*2/ ;    // set declaration
Parameter a(i) ;  // parameter declaration

Here the character sequence // serves as the end-of-line-comment indicator.
Attention

It is not allowed to reset the end-of-line comment symbol to the current end-of-line comment symbol. This would cause an compilation error as in the following example:

```
$eolCom //
$eolCom //
```

Some end of line character settings can cause confusion. The widely used end of line character sequence `//` is also legal GAMS syntax in `put` statement to indicate two line breaks:

```
file fx; put fx;
put 'first line' // 'second line' //;
$eolCom //
put 'third line' // 'fourth line';
```

results in a put file with the following content:

```
first line
second line
third line
```

This can also confuse syntax highlighting in editors (or on this web page). Other popular end of line characters like `#` and `@` are also used as GAMS syntax, see [Controlling the Cursor On a Page](#).

See also section [Comments](#) for more about comments in GAMS.

**$error**

**Syntax:**

```
$error [text]
```

This option will issue a compilation error and will continue with the next line.

**Example:**

```
$if not exist myfile
$error File myfile not found - will continue anyway
```

Note that the first line checks if the file `myfile` exists. If the file does not exist, it will generate an error with the comment `File myfile not found - will continue anyway` and then the compilation will continue with the next line.

See also `$abort`, `$exit`, `$terminate`, and `$stop`.

**$escape**

**Syntax:**

```
$escape character
```
This option allows users to work with text sequences containing % without substitution.

This causes all subsequent commands of the form %symbol% to not have parameter substitution done for them. As a consequence, no parameter substitutions are performed in GAMS statements (mostly useful in display and put statements) and the outcome of such statements where %symbol% is used is just %symbol%.

Note that the effect of the option $escape may be reversed with the option $escape %.

Example:

```gams
$set tt DOIT
file it; put it;

display "first %tt%";
display "second %&tt%&";
put "display one ", "%system.date%" /;
put "display two " "%&system.date%&"/;

$escape &
display "third %tt%";
display "fourth %&tt%&";
put "display third ", "%system.date%" /;
put "display fourth " "%&system.date%&"/;

$escape %
display "fifth %tt%";
display "sixth %&tt%&";
put "display fifth ", "%system.date%" /;
put "display sixth " "%&system.date%&"/;
```

The output generated by the display statements follows:

```
---- 6 first DOIT
---- 7 second %&tt%&
---- 12 third DOIT
---- 13 fourth %tt%
---- 18 fifth DOIT
---- 19 sixth %&tt%&
```

The file it.put will contain the following lines:

```
display one 08/10/17
display two %&system.date%&
display third 08/10/17
display fourth %system.date%
display fifth 08/10/17
display sixth %&system.date%&
```

Note that this option was introduced to facilitate writing GAMS code (or command.com/cmd.exe batch scripts) from GAMS including unsubstituted compile-time variables. Text can also be written at compile-time without parameter substitution via option $on/offEchoV and at run-time via $on/offPutV.
Note

In GAMS the escape character follows the character (%) that needs to be escaped. In many other languages the escape character precedes the to be escaped character.

$eval

Syntax:

$eval VARNAME expression

This option evaluates a numerical expression at compile time and places it into a scoped compile-time variable. In turn the option $ifE may be used to do numeric testing on the value of this variable.

VARNAME is the name of a compile-time variable and expression is an expression that consists of constants, functions, operators and other compile-time variables with numerical values. Note that no whitespace is allowed in the expression which can be overcome by additional parentheses.

Example:

$eval b1 ifthen(uniform(0,1)<0.5,0,1)
$eval b2 ifthen(uniform(0,1)<0.5,0,1)
$eval b3 (%b1%)xor(%b2%)
$log b1=%b1% b2=%b2% b1 xor b2=%b3%

The first two lines use the uniform function to generate a random number between 0 and 1 and assign 0 if this number is less than 0.5 otherwise 1 via the ifthen function to the scoped compile-time variable b1 and b1. In the third line we apply the logical xor operator to b1 and b2 and store the result in b3. The parentheses are required because the more natural expression %b1% xor %b2% contains spaces. In the forth line we print the values and result to the log.

b1=1 b2=1 b1 xor b2=0

The expression are evaluated using IEEE nonstop arithmetic, so no evaluation errors are triggered as demonstrated in the following example:

$eval OneDividedByZero 1/0
$log 1/0=%OneDividedByZero%

This produces the following log:

1/0=+INF

The $eval and related dollar control options give access to a reduced set of GAMS functions: abs, card, ceil, cos, errorlevel, exp, fact, floor, frac, gamsrelease, gamsversion, gday, gdow, ghour, gleap, gmillsec, gminute, gmonth, gsecond, gyear, ifthen, jdate, jnow, jobhandle, jobkill, jobstatus, jobterminate, jstart, jtime, log, log10, log2, max, min, mod, numcores, pi, power, round, sameas, sign, sin, sleep, sqrt, sqrt, tan, trunc, and uniform. The available operators are: +, -, *, /, ** and even ^ (integer power) which is not available in regular GAMS expression and requires the use of the function ipower. The comparison relations are <, >, <=, >=, <>, and =. The logical operators are not, and, or, xor, imp, and eqv.

The expression also allows the use of dollar on the right. In the following example we replace the ifthen function by a dollar one the right:
Moreover, the `$eval` has access to data available at compile time. The expression can access the value of scalars and for other symbols we can use the `card` function to access the cardinality (at this point) of the symbol. Here is an example:

```gams
Scalar ac 'Avogadro constant' / 6.0221409e+23 /
$eval log_ac round(log10(ac))
$log round(log10(ac))=%log_ac%
Set d / d0*d%log_ac% /
$eval card_d card(d)
$log card(d)=%card_d%
```

Access to individual records of symbols is not possible. The embedded code facility allows access to symbol records at compile time.

The user has the possibility to hide some of the GAMS functions by symbols with the same name. In this case, the GAMS function can still be accessed, but needs to be prefixed with `system`. as in the following example:

```gams
Scalar pi /3/;
$eval myPi pi
$eval GAMSPi system.pi
$log myPi = %myPi%
$log GAMSPi = %GAMSPi%
```

This is the resulting output in the log:

```plaintext
myPi = 3
GAMSPi = 3.14159265358979
```

See also `$eval.Set`, `$evalGlobal`, `$evalLocal`, `$ifE`, and `$set`.

$eval.Set

**Syntax:**

```
```

This option evaluates an attribute of a GAMS set at compile time and places it into a scoped compile-time variable. `VARNAME` is the name of a compile-time variable and `SETID` is the identifier of a GAMS set. The following attributes are allowed in this context:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[First OR Last]TE</td>
<td>Explanatory text of the first or last element of the set SETID</td>
</tr>
<tr>
<td>[First OR Last]TL</td>
<td>Label of the first or last element of the set SETID</td>
</tr>
<tr>
<td>[First OR Last]TN</td>
<td>Set identifier with with element labels of the first or last element of the set SETID</td>
</tr>
<tr>
<td>TS</td>
<td>Explanatory text of the set SETID. This is also the default, if no attribute is specified</td>
</tr>
</tbody>
</table>
Note

- **First??** and ?? are synonyms in the table above. It always accesses the first element of the referenced set.
- **Last??** always accesses the last element of the referenced set.
- For **Singleton Sets** First??, Last?? and ?? are synonyms.

**Example:**

Singleton Set h Greeting / Hello , 'Welcome' /
Set p Person / Mr.President , 'Male'
Mrs.Chancellor , 'Female' /

$eval.Set X h.TE
$log %X%
$eval.Set X p.lastTL
$log %X%
$eval.Set X p.FirstTN
$log %X%

This will generate the following in the log:

Welcome
Mrs.Chancellor
p('Mr','President')

See also $eval, $evalGlobal.Set, $evalLocal.Set, and $set.

$evalGlobal

**Syntax:**

$evalGlobal VARNAME expression

This option evaluates a numerical expression at compile time and places it into a global compile-time variable. The syntax and behavior otherwise is identical to $eval.

$evalGlobal.Set

**Syntax:**


This option evaluates an attribute of a GAMS set at compile time and places it into a global compile-time variable. The syntax and behavior otherwise is identical to $eval.Set.

$evalLocal

**Syntax:**

$evalLocal VARNAME expression

This option evaluates a numerical expression at compile time and places it into a local compile-time variable. The syntax and behavior otherwise is identical to $eval.

$evalLocal.Set

**Syntax:**
$evalLocal.Set VARNAME SETID[.][First|Last]TE[.][First|Last]TL[.][First|Last]TN|TS

This option evaluates an attribute of a GAMS set at compile time and places it into a local compile-time variable. The syntax and behavior otherwise is identical to $eval.Set.

$exit

Syntax:

$exit

This option will cause the compiler to exit (stop reading) from the current file. This is equivalent to having reached the end of file.

Example:

Scalar a ;
a = 5 ;
display a ;
$exit
a = a+5 ;
display a ;

Note that the lines following the option $exit will not be compiled.

Observe that there is a difference to the dollar control option $stop. If there is only one input file, $stop and $exit will have the same effect. If the option $exit occurs within an include file, it acts like an end-of-file on the include file. However, if the option $stop occurs within an include file, GAMS will stop reading all input.

See also $abort, $error, $terminate, and $stop.

$expose

Syntax:

$expose all | ident1 ident2 ...

This option removes all privacy restrictions from identifiers.

With explicit identifiers the privacy restrictions are removed only for the listed identifiers and with all the restrictions are removed for all identifiers. The privacy restrictions may be set with the dollar control options $hide or $protect. Note that a special license file is needed for this feature to work and that the expose only takes effect in subsequent restart files. For further information, see chapter Secure Work Files.

$funcLibIn

Syntax:

$FuncLibIn InternalLibName ExternalLibName

This makes extrinsic function libraries available to a model. InternalLibName is the internal name of the library in the GAMS code and ExternalLibName is the name of the shared library in the file system. See Using Function Libraries for more information.

$gdxIn

Syntax:
$gdxIn [GDXFileName]

This option is used in a sequence to load specified items from a GDX file. Here GDXFileName denotes the name of the GDX file (with or without the extension .gdx) and the command opens the specified GDX file for reading. The use of $gdxIn without a file name closes the currently open GDX file. The command is used in conjunction with the option $load or one of its variants.

Example:

set i,j; parameters a(i), b(j), d(i,j), f;
$gdxIn mydata.gdx
$load i j a b d f
$gdxIn

See also $load, and $gdxOut.

$gdxLoad

Syntax:

$gdxLoad GDXFileName [sym1[,] sym2=gdxSym2[,] sym3=[=]gdxSym3[.dimI][,] ...]

This option loads symbols from a specified GDX file. It basically combines $gdxIn and $load in one call. This can be particularly useful for $[bat|lib]include files when you want to be sure, to not interfere with potentially previously opened GDX files. For example, a $load after a "$gdxLoad" will still load the symbols from the GDX file opened before:

$gdxIn A.gdx
$load a b c
$gdxLoad B.gdx d
* The following $load gets the symbols e, f, and g from GDX file A.gdx
$load e f g

See also $[on][off]Filtered, $[on][off]Multi, and $onMultiR.

$gdxOut

Syntax:

$gdxOut [GDXFileName]

This option is used in a sequence to unload specified items to a GDX file at compile time. Here GDXFileName denotes the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for writing. The use of $gdxOut without a file name closes the currently open output GDX file and initiates the actual writing of the sequence of symbols. The command is used in conjunction with the dollar control option $unLoad.

Example:

set i /i1*i3/; parameters a(i) /i1 3, i2 87, i3 1/;
$gdxOut mydata.gdx
$unLoad i a
$gdxOut

See also $unLoad, and $gdxIn.

$gdxUnload
Syntax:

$gdxUnload GDXFileName [sym1[, sym2=gdxSym2[, ...]]

This option unloads symbols from a specified GDX file. It basically combines $gdxOut and $unload in one call. This can be particularly useful for $batlibinclude files when you want to be sure, to not interfere with potentially previously opened GDX files. For example, a $unload after a "$gdxUnload" will still unload the symbols to the GDX file opened before:

$gdxOut A.gdx
$unload a b c
$gdxUnload B.gdx d
* The following $unload puts the symbols e, f, and g into GDX file A.gdx
$unload e f g

$goto

Syntax:

$goto id
$label id

This option will cause GAMS to search for a line starting with $label id and then continue reading from there. This option can be used to skip over or repeat sections of the input files. In $batinclude files the target labels or label arguments can be passed as parameters because of the manner in which parameter substitution occurs in such files. In order to avoid infinite loops, jumps to the same label are restricted to a maximum of 100 times by default. This maximum may be changed with the option $maxGoto.

Example:

Scalar a ;
a = 5;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;

Note that GAMS will continue from line $label next after reading line $goto next. Observe that all lines in between are ignored. Therefore the final value of a in the example above will be 15.

Attention

The lines $goto and $label have to be in the same file. If the target label is not found in the current file an error will be issued.

See also $label, $maxGoto.

$hidden

Syntax:

$hidden text
A line starting with this option will be ignored and will not be echoed to the listing file. This
option is used to enter information only relevant to the person manipulating the file.

Example:

$hidden You need to edit the following lines if you want to:
$hidden
$hidden  1. Change form a to b
$hidden  2. Expand the set

The lines above serve as comments to the person who wrote the file. However, these comments
will not be visible in the listing file and are therefore hidden from view.

Note

This option is particularly useful when the input file is encrypted.

$hiddenCall

Syntax:

$hiddenCall [=}command

This option does the same as $call but the statement is neither shown on the log nor the
listing file.

$hiddenCall.Async[NC]

Syntax:

$hiddenCall.Async[NC] command

This option does the same as $call.Async[NC] but the statement is neither shown on the log
nor the listing file.

$hiddenCall.checkErrorLevel

Syntax:

$hiddenCall.checkErrorLevel [=}command

This option does the same as $call.checkErrorLevel but the statement is neither shown on the
log nor the listing file.

$hide

Syntax:

$hide all | ident1 ident2 ...

This option hides identifiers so they cannot be displayed or computed, but they may still be
used in model calculations (i.e. commands when the solve statement is executed).

With explicit identifiers the listed identifiers are hidden and with all all identifiers are hidden.
These restrictions may be removed with the dollar control options expose or purge. Note that
a special license file is needed for this feature to work.

For further information, see chapter Secure Work Files.
$\textit{if}$

\textbf{Syntax:}

\texttt{$\textit{if}$ [not] conditional\_expression new\_input\_line}

This dollar control option provides the greatest amount of control over conditional processing of the input file(s).

For more information on the \texttt{conditional\_expression} allowed, details on the \texttt{new\_input\_line} and examples, see section \textit{Conditional Compilation} below.

See also $\textit{ifE}$, $\textit{ifI}$, $\textit{ifThen}$.

$\textit{ifE}$

\textbf{Syntax:}

\texttt{$\textit{ifE}$ [not] conditional\_expression new\_input\_line}

This dollar control option does the same as the option $\textit{if}$ but allows constant expression evaluation. The \texttt{conditional\_expression} may take two different forms:

\begin{align*}
\text{expr1} &= \text{expr2} & \text{TRUE} & \text{if } (\text{expr1}-\text{expr2})/(1+\text{abs(}\text{expr2})) < 1\text{e-12} \\
\text{expr} &= \text{TRUE} & \text{if } \text{expr1} < 0
\end{align*}

\textbf{Example:}

Scalar \texttt{a};
\begin{align*}
&$\textit{ifE} \ (\log2(16)^2)=16 \quad a=0; \ \text{display a;} \\
&$\textit{ifE} \ \log2(16)^2 == 16 \quad a=1; \ \text{display a;} \\
&$\textit{ifE} \ \text{NOT round}(\log2(16)^2-16) \quad a=2; \ \text{display a;} \\
&$\textit{ifE} \ \text{round}(\log2(16)^2-16) \quad a=3; \ \text{display a;} \\
&$\textit{ifE} \ \text{round}(\log2(16)^2-17) \quad a=4; \ \text{display a;}
\end{align*}

This will create the following output:

\begin{align*}
{\text{----}} & \quad 3 \text{ PARAMETER a} & \quad = \quad 1.000 \\
{\text{----}} & \quad 4 \text{ PARAMETER a} & \quad = \quad 2.000 \\
{\text{----}} & \quad 6 \text{ PARAMETER a} & \quad = \quad 4.000
\end{align*}

See also $\textit{if}$ and section \textit{Conditional Compilation}.

$\textit{ifI}$

\textbf{Syntax:}

\texttt{$\textit{ifI}$ [not] conditional\_expression new\_input\_line}

This option is working like the option $\textit{if}$. The only difference is that $\textit{if}$ makes comparisons involving text in a case sensitive fashion while $\textit{ifI}$ is case insensitive.

See also $\textit{if}$ and section \textit{Conditional Compilation}.

$\textit{ifThen}$

\textbf{Syntax:}

\texttt{$\textit{ifThen}$ [not] conditional\_expression new\_input\_line}

This option is working like the option $\textit{if}$. The only difference is that $\textit{if}$ makes comparisons involving text in a case sensitive fashion while $\textit{ifThen}$ is case insensitive.

See also $\textit{if}$ and section \textit{Conditional Compilation}. 
$ifThen[E|I] \ cond
... 
{ $elsif[E|I] \ cond
... 
} 
[ $else 
... 
] 
$endif

This option is a form of the option $if that controls whether a number of statements are active. The syntax for the condition is generally the same as for the option $if. Like $if, it is case sensitive. Often it is followed by one or more of the following dollar control options: $else, $elsif, $elsifI, $elsifE. The option $ifThen must be matched with the option $endif that marks the end of the construct. An example is given in section Conditional Compilation with $ifThen and $else.

Note that users may add a tag to the $ifThen and $endif. For example, $ifThen.tagOne has to match with $endif.tagOne.

Example:

$ifThen.one \ x == y 
display "it1";
$elsif.one \ a == a 
display "it2";
$ifThen.two \ c == c 
display "it3";
$endif.two
$elsif.one \ b == b 
display "it4";
$endif.one

The resulting listing file will contain the following lines:

---- 2 it2
---- 4 it3

Note that the first condition (x == y) is obviously not true and the fourth condition (b == b) is not tested because the second condition (a == a) was already true.

See also $if, $ifThen, $ifThenI, $else, $elsif and section Conditional Compilation.

$ifThenE

Syntax:

$ifThen[E|I] \ cond
... 
{ $elsif[E|I] \ cond
... 
} 
[ $else 
... 
] 
$endif

This option does the same as the option $ifThen but evaluates numerical values of the control variables.

See also $ifThen and section Conditional Compilation.
$ifThenI

Syntax:

$ifThen[E|I] cond
...
{ $elseif[E|I] cond
...
}
[ $else
...
] $endif

This option does the same as the option $ifThen but it is case insensitive.

See also $ifThen and section Conditional Compilation.

$include

Syntax:

$include external_file

This option inserts the contents of a specified text file at the location of the call. Note that text files that have been compressed with Posix utility gzip or have been encrypted with tool ENDECRYPT are also allowed. External_file is the name of the file that is included. It can be quoted or unquoted. Note that include files may be nested.

The include file names are processed in the same way like the input file. The names are expanded using the working directory. If the file cannot be found and no extension is given, the standard GAMS input extension is tried. However, if an incomplete path is given, the file name is completed using the include directory. By default, the library include directory is set to the working directory. The default directory search path may be extended with the command line parameter InputDir.

Note that the start of the include file is marked and the include file is echoed to the echo print in the listing file. This reference to the include file may be omitted by using the option $offInclude.

Example:

$include myfile
$include "myfile"

Both statements above are equivalent and the search order for the include file is as follows:

1. myfile in current working directory
2. myfile.gms in current working directory
3. myfile and myfile.gms (in that order) in directories specified by the command line parameter InputDir.
Attention

The current settings of the dollar control options are passed on to the lower level include files. However, the dollar control options set in the lower level include file are passed on to the parent file only if the option $onGlobal is set.

Note that details on the compilation output of include files are given in section The Include File Summary.

See also $batInclude, $libInclude, $sysInclude.

$inlineCom (/ * */)

Syntax:

$inlineCom char[char] char[char]

This option redefines and activates the in-line comment symbols. These symbols are placed at the beginning and the end of the in-line comment and are one character or a two character sequence at the beginning and the end. By default, the system is initialized to '/*' and '*/*', but is not active. The option $onInline is used to activate the in-line comments. If $inlineCom is used, $onInline is set automatically.

Example:

$inlineCom {{ }}
Set {{ this is an inline comment }} i / 1*2 /;

Note that the character pairs {{ }} serve as the indicator for in-line comments.

Attention

It is not allowed to reset the option $inlineCom to the current symbol for in-line comments. This would cause a compilation error as in the following example:

$inlinecom {{ }}
$inlinecom {{ }}

Note

The option $onNestCom enables the use of nested comments.

See also section Comments.

$kill

Syntax:

$kill ident {ident}

This option removes all data for the identifiers ident, only the type and dimension are retained (this means that these identifiers will be declared but not defined anymore). Note that only the data of the following data types may be removed: sets, parameters, variables and equations. Note further that the data removal is carried out during compile time and not when the GAMS program executes.

Example:
Set i / 1*20 /;
Scalar a /2/;
$kill i a

Note that the effect of the third line above is that all data from \(a\) and \(i\) is removed, so the set \(i\) and the scalar \(a\) are declared, but not initialized or assigned to. Note that after \(i\) and \(a\) have been killed, a display statement for them will trigger an error. However, new data may be assigned to identifiers that were previously killed. Thus the following statements are valid if appended to the code above:

Set i / i1 *i3 /;
a = 7;

Observe that this option needs to be distinguished from the dollar control option $clear, that resets the data to the default values.

$label

Syntax:

$goto id
$label id

This option marks a line to be jumped to by a dollar control option $goto. Any number of labels may be used in files and not all of them need to be referenced. Re-declaration of a label identifier will not generate an error and only the first occurrence encountered by the GAMS compiler will be used for future $goto references.

Example:

Scalar a ;
a = 5 ;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;

When GAMS reaches the line $goto next, it continues from the line $label next. All lines in between are ignored. Therefore in the example above, the final value of \(a\) is 15.

Attention

If several dollar control options appear in one line and label is one of them, then label must be listed first.

See also $goto, $maxGoto.

$libInclude

Syntax:

$libInclude external_file {arg}
This option is mostly equivalent to the option $batInclude. However, if an incomplete path is given, the file name is completed using the library include directory. By default, the library include directory is set to the directory inclib in the GAMS standard locations. Note that the list of default directories searched may be prepended with the command line parameter libIncDir.

Example:

$libInclude abc x y

This call will first look for the include file inclib/abc in the standard locations. If this file does not exist there GAMS will look for the file inclib/abc.gms in the standard locations. The arguments x and y are passed on to the include file and are interpreted as explained in the detailed description of the option $batInclude.

See also $include, $batInclude, $sysInclude.

$lines

Syntax:

$lines n

This option starts a new page in the listing file if less than n lines are available on the current page.

Example:

$hidden Never split the first few lines of the following table
$lines 5
Table io(i,j) Transaction matrix
...
;

This will ensure that if there are less than five lines available on the current page in the listing file before the next statement (in this case, the table statement) is echoed to it, the contents of this statement are echoed to a new page.

$load

Syntax:

$load [sym1[,] sym2=gdxSym2[,] sym3<=[=]gdxSym3[,].dimI][,] ...]

This option is preceded and succeeded by the option $gdxIn that open the GDX file for reading. The option $load loads specified items from the GDX file. Note that more than one instance of $load may occur. A listing of the GDX file contents will be created if the option $load is not followed by arguments.

Examples

Consider the following example, where transsol is the GDX file of the transportation model [TRANSPORT]

$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i j a b d f
$gdxIn
A comma between the symbols is optional. The follow example works identically:

```gams
$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i, j, a, b, d, f
$gdxIn
```

The `$load` without any arguments produces a table of contents of the GDX container in the listing file:

```
Content of GDX C:\Users\default\Documents\gamsdir\projdir\transsol.gdx
5 UELs

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Dim</th>
<th>Count</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set</td>
<td>1</td>
<td>2</td>
<td>i canning plants</td>
</tr>
<tr>
<td>2</td>
<td>Set</td>
<td>1</td>
<td>3</td>
<td>j markets</td>
</tr>
<tr>
<td>3</td>
<td>Parameter</td>
<td>1</td>
<td>2</td>
<td>a(i) capacity of plant i in cases</td>
</tr>
<tr>
<td>4</td>
<td>Parameter</td>
<td>1</td>
<td>3</td>
<td>b(j) demand at market j in cases</td>
</tr>
<tr>
<td>5</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>d(i,j) distance in thousands of miles</td>
</tr>
<tr>
<td>6</td>
<td>Parameter</td>
<td>0</td>
<td>1</td>
<td>f freight in dollars per case per thousand miles</td>
</tr>
<tr>
<td>7</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>c(i,j) transport cost in thousands of dollars per case</td>
</tr>
<tr>
<td>8</td>
<td>Variable</td>
<td>2</td>
<td>6</td>
<td>x(i,j) shipment quantities in cases</td>
</tr>
<tr>
<td>9</td>
<td>Variable</td>
<td>0</td>
<td>1</td>
<td>z total transportation costs in thousands of dollars</td>
</tr>
<tr>
<td>10</td>
<td>Equation</td>
<td>0</td>
<td>1</td>
<td>cost define objective function</td>
</tr>
<tr>
<td>11</td>
<td>Equation</td>
<td>1</td>
<td>2</td>
<td>supply(i) observe supply limit at plant i</td>
</tr>
<tr>
<td>12</td>
<td>Equation</td>
<td>1</td>
<td>3</td>
<td>demand(j) satisfy demand at market j</td>
</tr>
</tbody>
</table>
```

Symbols may be loaded with new names with the following syntax: `$load i=gdx_i j=j.gdx`. The universal set may be loaded using `$load uni=*`.

```gams
$gdxIn transsol
Sets i, jj, uni; Parameters a(i), bb(jj), d(i,jj), f;
$load i jj=j uni=* a bb=b d f
$gdxIn
display uni;
```

This results in a display of all used labels:

```
---- 5 SET uni
Seattle , San-Diego, New-York , Chicago , Topeka
```

The syntax `sym<=[=]GDXSym.[dimI]` allows to load a one dimensional set from a symbol in the GDX file that has even a higher dimensionality. GAMS tries to find the set `sym` as a domain in the symbol `GDXSym` and uses the labels from this index position (with `<` the first domain set from the right and with `<=` from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix `.dimI` allows to pick a fixed index position.

In the following we work with a GDX file created by the following code:

```gams
set i / i1*i3 /, ii(i,i) / i1.i2, i2.i3 /;
execute_unloadDI 'ii', i, ii;
```

Now use use this GDX file to load the first and second index from `ii`:

```gams
set i, ii;
$gdxIn ii
* Load first index from ii as i
$load i<=ii i1<ii.dimI
display i, ii;
```
the display lists all labels from the first index of ii:

```
---- 5 SET i Domain loaded from ii position 1
 i1, i2
---- 5 SET i1 Domain loaded from ii position 1
 i1, i2
```

Now we match from the right and get the second index of ii:

```
set i, i2;
$gdxIn ii
* Load second index from ii as i
$load i<ii i2<ii.dim2
display i, i2;
```

The resulting listing file will contain the following lines:

```
---- 5 SET i Domain loaded from ii position 2
 i2, i3
---- 5 SET i2 Domain loaded from ii position 2
 i2, i3
```

This type of projection loading can be useful to extract the domain sets from a single parameter that is stored in a GDX file:

```
set i,j,k; parameter data(i,j,k);
$gdxIn data
$load i<data.dim1 j<data.dim2 k<data.dim3 data
```

Attention

Loading an item that was already initialized will cause a compilation error.

For example, the following code snippet will cause a compilation error:

```
Set j / 1*5 /;
$gdxIn transsol
$load j
$gdxIn
```

Note that GAMS offers variants of `$load` that do not cause a compilation error in such a case: `$loadM` and `$loadR`.

**$loadDC**

**Syntax:**

```
$loadDC [sym1[, sym2=gdxSym2[, sym3<=[=]gdxSym3[.dim1][,] ...]]
```

This option is an alternative form of `$load`. It performs domain checking when items are loaded. Any domain violations will be reported and flagged as compilation errors. This is basically the opposite of `$loadFiltered`. All other features are the same as discussed under `$load`.

**Example:** Consider the following example where `transsol` is the GDX file of the transportation model `[TRNSPORT]`. 
Set i, j;
Parameter b(i), a(j);
$gdxIn transsol
$load i b
$loadDC j a
$gdxIn

Note that in contrast to the example above, the parameter a is indexed over the set i and the parameter b is indexed over the set j in the file transsol. While $load i b does not generate an error and b is just empty, the option $loadDC j a triggers a domain violation error because in transsol a is indexed over i and produces a list of errors in the listing file:

---  LOAD  a = 3:a
**** Unique domain errors for symbol a
  Dim  Elements
   1  seattle, san-diego

   5  $loadDC j a
****  $649

$loadDCM

Syntax:

$loadDCM [sym1[, sym2=gdxSym2[, sym3<=[=]gdxSym3[, dim1][,] ...]]

This option combines the functionality of merging as in $loadM and domain checking as in $loadDC.

Example:

Consider the following example where transsol is the GDX file of the transportation model [TRANSPORT].

Set i, uni 'all labels';
Parameter abFail(i), ab(uni) 'capacity and demand';
$gdxIn transsol
$load i abFail=a
$loadDCM abFail=b
$loadDCM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;

Here we try to merge parameters a and b together into one parameter. The first attempt (to merge it into parameter abFail) would fail because of line 5 and result into a domain violation report as described with dollar control option $loadDC. In the second attempt we first merge the sets i and j into set uni and then merge the parameters a and b into ab. If one comments line 5 the resulting display looks as follows:

---  8 SET uni all labels
    seattle, san-diego, new-york, chicago, topeka

---  8 PARAMETER ab capacity and demand
    seattle 350.000, san-diego 600.000, new-york 325.000
    chicago 300.000, topeka 275.000

$loadDCR

Syntax:
$\text{loadDCR} \ [\text{sym1[,] sym2=gdxSym2[,] sym3=[=]gdxSym3[.dimI[,] \ldots]}$

This option combines the functionality of replacing data as in $\text{loadR}$ and domain checking as in $\text{loadDC}$.

Example:

Consider the following example where transsol is the GDX file of the transportation model [TRNSPORT].

Set uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$\text{gdxIn transsol}$
$\text{loadM uni=i uni=j ab=a}$
$\text{loadDCR ab=b}$
$\text{gdxIn}$
display uni, ab;

Here we try to read twice into the parameter $ab$. First GDX symbol $a$ and $b$ are read into $ab$. GDX symbol $b$ is read with replace and hence the parameter $ab$ contains the elements of $b$ only.

$\text{loadFiltered}$

Syntax:

$\text{loadFiltered} \ [\text{sym1[,] sym2=gdxSym2[,] sym3=[=]gdxSym3[.dimI[,] \ldots]}$

This option is an alternative form of $\text{load}$, in particular, it works like $\text{load}$ while $\text{SonFiltered}$ is set. That means, when items are loaded, domain violations are not flagged as compilation errors, but items violating the domain definition are just filtered out. This is basically the opposite of $\text{loadDC}$. All other features are the same as discussed under $\text{load}$.

Example: Consider the following example where transsol is the GDX file of the transportation model [TRNSPORT].

Set i / seattle /
    j / new-york, chicago /;
Parameter a(i), b(j);
$\text{gdxIn transsol}$
$\text{loadFiltered a}$
$\text{loadDC} \ b$
$\text{gdxIn}$

While $\text{loadFiltered a}$ does not generate an error, but just ignores the data that does not match the defined domain, $\text{loadDC} \ b$ triggers a domain violation error.

See also $\text{loadFilteredM}$ and $\text{loadFilteredR}$.

$\text{loadFilteredM}$

Syntax:

$\text{loadFilteredM} \ [\text{sym1[,] sym2=gdxSym2[,] sym3=[=]gdxSym3[.dimI[,] \ldots]}$
This option combines the functionality of \textit{merging} as in \texttt{$loadM} and domain filtering as in \texttt{$loadFilterd}.

\textbf{Example:}

Consider the following example where \texttt{transsol} is the GDX file of the transportation model [\texttt{TRNSPORT}].

\begin{verbatim}
Set i, uni 'all labels';
Parameter abFilter(i), ab(uni) 'capacity and demand';
$gdxIn transsol
$load i abFilter=a
$loadFilteredM abFilter=b
$loadFilteredM uni=i uni=j ab=a ab=b
$gdxIn
Display uni, ab, abFilter;
\end{verbatim}

Here we try to merge parameters \texttt{a} and \texttt{b} together into one parameter. The first attempt (to merge it into parameter \texttt{abFilter}) does not add anything, since the elements in \texttt{b} do not match the defined domain \texttt{i} (compare \texttt{$loadFilterd}). In the second attempt we first merge the sets \texttt{i} and \texttt{j} into set \texttt{uni} and then merge the parameters \texttt{a} and \texttt{b} into \texttt{ab}. The \texttt{Display} statement produces the following output:

\begin{verbatim}
---- 31 SET uni all labels
  seattle, san-diego, new-york, chicago, topeka

---- 31 PARAMETER ab capacity and demand
  seattle 350.000, san-diego 600.000, new-york 325.000, chicago 300.000, topeka

---- 31 PARAMETER abFilter capacity of plant i in cases
  seattle 350.000, san-diego 600.000
\end{verbatim}

\texttt{$loadFilteredR}

\textbf{Syntax:}

\texttt{$loadFilteredR [sym1[,] sym2=gdxSym2[,] sym3=[\texttt{=}gdxSym3[].dimI][][,] \ldots]}

This option combines the functionality of \textit{replacing} data as in \texttt{$loadR} and domain filtering as in \texttt{$loadFilterd}.

\textbf{Example:}

Consider the following example where \texttt{transsol} is the GDX file of the transportation model [\texttt{TRNSPORT}].

\begin{verbatim}
Set uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadFilteredM uni=i uni=j ab=a
$loadFilteredR ab=b
$gdxIn
Display uni, ab;
\end{verbatim}
Here we try to read twice into the parameter ab. First GDX symbol a and b are read into ab. GDX symbol b is read with replace and hence the parameter ab contains the elements of b only:

---- 31 SET uni all labels
seattle, san-diego, new-york, chicago, topeka

---- 31 PARAMETER ab capacity and demand
new-york 325.000, chicago 300.000, topeka 275.000

$loadM
Syntax:
$loadM [sym1[, sym2=gdxSym2[, sym3<=[gdxSym3[dimI][,] ...]]]

This option is an alternative form of $load. Instead of replacing an item or causing a symbol redefined error if the item was already initialized it merges the contents. Records that would result in domain violations will be ignored.

Example:
Consider the following example where transsol is the GDX file of the transportation model [TRNSPORT].

Set i, uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;

Here we merge parameters a and b together into one parameter ab. We first merge the sets i and j into set uni and then merge the parameters a and b into ab. The resulting display looks as follows:

---- 6 SET uni all labels
seattle, san-diego, new-york, chicago, topeka

---- 6 PARAMETER ab capacity and demand
seattle 350.000, san-diego 600.000, new-york 325.000
chicago 300.000, topeka 275.000

$loadR
Syntax:
$loadR [sym1[, sym2=gdxSym2[, sym3<=[gdxSym3[dimI][,] ...]]]

This option is a variant of the option $load. With $loadR we can have multiple loads into the same symbols and the data stored in GAMS will be replaced with the one from the GDX container.

Example:
Consider the following example, where transsol is the GDX file of the transportation model [TRNSPORT]:

--- 31 SET uni all labels
seattle, san-diego, new-york, chicago, topeka

--- 31 PARAMETER ab capacity and demand
new-york 325.000, chicago 300.000, topeka 275.000

$loadM
Sets i / 1*3 /  
  j / 1*2 /;
$gdxIn transsol
$loadR i j
$gdxIn
display i, j;

The resulting listing file will contain the following lines:

---- 6 SET i canning plants
Seattle , San-Diego

---- 6 SET j markets
New-York, Chicago , Topeka

$log

Syntax:

$log text

This option will send a message text to the log file. Recall that by default, the log file is the console. The default log file may be reset with the command line parameters logOption and logFile.

Attention

• Leading blanks are ignored when the text is written out to the log file as a result of using the $log option.
• All special % symbols will be substituted before the text passed through the $log option is sent to the log file.

Example:

$log
$log The following message will be written to the log file
$log with leading blanks ignored. All special % symbols will
$log be substituted before this text is sent to the log file.
$log This was line %system.incLine% of file %system.incName%
$log

The log file that results by running the lines above will contain the following lines:

The following message will be written to the log file
with leading blanks ignored. All special % symbols will
be substituted before this text is sent to the log file.
This was line 5 of file C:\tmp\logTest.gms

Note that %system.incLine% is replaced by 5 which is the line number where the string replacement was requested. Note further that %system.incName% is substituted with the name of the file completed with the absolute path. Observe that the leading blanks on the second line of the example are ignored.

$macro

Syntax:

$macro name(arg1,arg2,arg3,...) macro_body
This option defines a macro in GAMS. Here \texttt{name} is the name of the macro, \texttt{arg1, arg2, arg3, ...} are the arguments and \texttt{macro body} defines what the macro should do. The macro names follow the rules for identifiers. The macro \texttt{name} cannot be used for other symbols. For further details and examples, see section Macros in GAMS below.

\texttt{\$maxCol \hspace{1em} (80001)}

\textbf{Syntax:}

\texttt{\$maxCol \hspace{0.5em} n}

This option restricts the valid range of input columns at the \textit{right} margin. Note that all input after column \texttt{n} is treated as comment, therefore it is ignored.

\textbf{Example:}

\texttt{\$maxCol 30}
\texttt{Set \hspace{1em} i / vienna, rome /; \hspace{1em} set definition}
\texttt{Scalar \hspace{0.5em} a / 2.3 /; \hspace{0.5em} scalar definition}

Observe that the text strings \texttt{set definition} and \texttt{scalar definition} are treated as comments and are ignored since they begin on or after column 31.

Any changes in the margins via \texttt{\$maxCol} or \texttt{\$minCol} will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option \texttt{\$minCol 20 \hspace{1em} \$maxCol 110} will trigger the following message:

\textbf{NEW MARGINS: 20–110}

\textbf{Note}

\begin{itemize}
  \item GAMS requires that the right margin set by \texttt{\$maxCol} is greater than 15.
  \item GAMS requires that the right margin set by \texttt{\$maxCol} is greater than the left margin set by \texttt{\$minCol}.
\end{itemize}

See also \texttt{\$on/offMargin} and section Comments.

\texttt{\$maxGoto 100}

\textbf{Syntax:}

\texttt{\$maxGoto \hspace{0.5em} n}

This option sets the maximum number of jumps to the same label and is used in the context of the options \texttt{\$goto} and \texttt{\$label}. Once the maximum number is reached a compilation error is triggered. Such a limit has been implemented to avoid infinite loops at compile time.

\textbf{Example:}

\texttt{Scalar \hspace{0.5em} a / 1 /;}
\texttt{\$maxGoto 5}
\texttt{\$label label1}
\texttt{a = a+10;}
\texttt{display a ;}
\texttt{\$goto label1}

Note that a compilation error is triggered if \texttt{\$goto label1} is called for the fifth time.

\texttt{\$minCol \hspace{1em} (1)
Syntax:

$\text{minCol } n$

This option restricts the valid range of input columns at the left margin. Note that all input before column $n$ is treated as comment, therefore it is ignored.

Example:

$\text{minCol 30}$

\begin{verbatim}
Set definition     Set  i / vienna, rome /;
Scalar definition  Scalar a / 2.3    /;
\end{verbatim}

Observe that the text strings set definition and scalar definition are treated as comments and are ignored since they are placed before column 30.

Any changes in the margins via the option $\text{maxCol}$ or $\text{minCol}$ will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option $\text{minCol 20 maxCol 110}$ will trigger the message:

NEW MARGINS: 20-110

Attention

GAMS requires that the left margin set by the option $\text{minCol}$ is smaller than the right margin set by the option $\text{maxCol}$.

See also $\text{on/offMargin}$ and section Comments.

$\text{[on][off]CheckErrorLevel (offCheckErrorLevel)}$

Syntax:

$\text{onCheckErrorLevel}$

$\text{offCheckErrorLevel}$

If the option $\text{onCheckErrorLevel}$ is active, the errorLevel is checked implicitly after $\text{call}$ and $\text{hiddenCall}$. A compilation error is triggered and the compilation is aborted if it is not 0. With $\text{offCheckErrorLevel}$, this is not done, but one can do it explicitly. So, with $\text{onCheckErrorLevel}$ $\text{call}$ behaves like $\text{call.checkErrorLevel}$ and $\text{hiddenCall}$ behaves like $\text{hiddenCall.checkErrorLevel}$.

$\text{[on][off]Delim (offDelim)}$

Syntax:

$\text{onDelim}$

$\text{offDelim}$

This option controls whether data in table statements may be entered in delimited format. As delimiter one can use ",", ",", or "TAB".

Example:
Sets  plant 'plant locations'  / NEWYORK, CHICAGO, LOSANGELES /  
       market 'demands'       / MIAMI, HOUSTON, PORTLAND    /;

Table dist(plant,market)  
$onDelim  
   ,MIAMI,HOUSTON,PORTLAND  
NEWYORK,1300,1800,1100  
CHICAGO,2200,1300,700  
LOSANGELES,3700,2400,2500  
$offDelim  
;  

Display dist;  

The resulting listing file will contain the following output:  

---- 12 PARAMETER dist  

<table>
<thead>
<tr>
<th></th>
<th>MIAMI</th>
<th>HOUSTON</th>
<th>PORTLAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEWYORK</td>
<td>1300.000</td>
<td>1800.000</td>
<td>1100.000</td>
</tr>
<tr>
<td>CHICAGO</td>
<td>2200.000</td>
<td>1300.000</td>
<td>700.000</td>
</tr>
<tr>
<td>LOSANGELES</td>
<td>3700.000</td>
<td>2400.000</td>
<td>2500.000</td>
</tr>
</tbody>
</table>

$[on][off]Digit  ($onDigit)  

Syntax:  

$onDigit  
$offDigit  

Historically, GAMS has run on widely different platforms where even the floating-point arithmetic varied. To ensure the precision specified in the GAMS input didn't exceed what could be stored and computed with as GAMS ran, we limited the precision (i.e. the number of significant digits used for numbers) that could be specified in the GAMS source. This limit could be relaxed by using the $offDigit control.  

Modern computers all support IEEE double-precision arithmetic, and GAMS uses this internally. It's a well-accepted and familiar standard, and the current behavior - to convert the decimal value in the GAMS source into the nearest double-precision value, regardless of the number of digits in the GAMS source - is what is generally expected. There is a limit (currently 40) on the number of digits treated as potentially significant: digits beyond this point are quietly treated as zero.  

The $offDigit control remains useful for changing how GAMS handles numbers with very large or small magnitude. With $onDigit, GAMS throws compiler warnings/errors for these values. With $offDigit, GAMS quietly converts them to UNDF or EPS, respectively.  

Example:  

scalar aboveTheLimit / 222e299 /;  
scalar tiny / 1.0e-275 /;  
scalar tinier / 1.0e-285 /;  
$offdigit  
scalar uberTheLimit / 222e299 /;  
scalar klein / 1.0e-275 /;  
scalar kleiner / 1.0e-285 /;
The resulting listing file will contain the following lines:

```
1 scalar aboveTheLimit / 222e299 /;
**** $106
2 scalar tiny / 1.0e-275 /;
**** $595
3 scalar tinier / 1.0e-285 /;
**** $594
5 scalar uberTheLimit / 222e299 /;
6 scalar klein / 1.0e-275 /;
7 scalar kleiner / 1.0e-285 /;
```

Error Messages

```
106 abs(number) out of range
abs(number) must be less than 1E300
   $offdigit can be used to replace large numbers with UNDF
594 Exponent very very small - rejected
   $offdigit can be used to replace with EPS
595 Exponent very small
   $offdigit can be used to quietly accept very small values
```

```
$[on][off]Dollar  ($offDollar)

Syntax:

$onDollar
$offDollar

This option controls the echoing of dollar control option lines in the listing file.

Example:

```
$hidden This line will not be displayed
$onDollar
$hidden This line will be displayed
$offDollar
$hidden This line will not be displayed
```

The compilation output of the resulting listing file will contain the following lines:

```
2  $onDollar
3  $hidden This line will be displayed
```

Note that all lines between the option $onDollar and the option $offDollar are echoed in the listing file. Note further that the effect of this option is immediate: the line $onDollar is echoed in the listing file, while the line $offDollar is not.

```
$[on][off]DotL ($offDotL)

Syntax:

$onDotL
$offDotL
```
This option activates or deactivates the automatic addition of the attribute .L to variables on the right-hand side of assignments and in put statement. It is most useful in the context of macros. For further information, see section Macros in GAMS below.

Example:

Consider the following code snippet which can be added to the end if the `TRNSPORT` model:

```gams
$onDotL
file fx; put fx;
loop((i,j),
   put / x(i,j)
);
$offDotL
file fx; put fx;
loop((i,j),
   put / x(i,j)
);
```

This is the generated output:

```
50.00
300.00
0.00
275.00
0.00
275.00
x('seattle','new-york')
x('seattle','chicago')
x('seattle','topeka')
x('san-diego','new-york')
x('san-diego','chicago')
x('san-diego','topeka')
```

First, we see the level for each x record. After setting `$offDotL` we are back at the default which means, that the .tn attribute is used implicitly.

$[on][off]DotScale ($offDotScale)

Syntax:

```
$onDotScale
$offDotScale
```

This option activates or deactivates the automatic addition of the attribute .Scale to variables and equations on the right-hand side of assignments and in put statements. As with on/offDotL It is most useful in the context of macros. For further information, see section Macros in GAMS below.

$[on][off]Echo[S][V][.tag]

Syntax:
This option is used to send one or more lines of text to an file external_file. The text and the file name may be quoted or unquoted. The external file is not closed until the end of the compilation or when the option $call$ or any variant of the option $include$ is encountered. Note that the redirection symbols $>$ and $>>$ have the usual meaning: $>$ creates a new file and writes to it or - in case there exists already a file with the respective name - overwrites the existing file and $>>$ appends to a file. Note further that parameter substitutions are permitted with $onEcho$. The option $onEcho$ has two more variants: $onEchoS$ and $onEchoV$. $onEchoS$ allows parameter substitutions like $onEcho$, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended S. The option $onEchoV$ does not allow parameter substitutions but writes the text verbatim.

Example:

```
$set it TEST
$onEchoS > externalfile1.txt
send %it% to external file
line 2 to send
$offEcho

$onEchoV > externalfile2.txt
send %it% to external file
line 2 to send
$offEcho
```

The externalfile1.txt will contain the following lines:

```
send TEST to external file
line 2 to send
```

The externalfile2.txt will contain these lines:

```
send %it% to external file
line 2 to send
```

Observe that in the first case %it% is substituted with TEST, but in the second case there is no substitution.

Note that by default the external file will be placed in the current working directory if there is no path specified.

One may add a tag to the $onEcho[S|V]$ and $offEcho$. This allows to use nested echo blocks like this:

```
$onEcho.outer > outerFile.gms
$onEcho.inner > innerFile.txt
Line 1 of text
Line 2 of text
$offEcho.inner
$offEcho.outer
```

The resulting file outerFile.gms looks like this:
4.41 Dollar Control Options

$onEcho.inner > innerFile.txt
Line 1 of text
Line 2 of text
$offEcho.inner

See also options $echo, and $echoN.

$[on][off]ECImplicitLoad  ($onECImplicitLoad)

Syntax:

$onECImplicitLoad
$offECImplicitLoad

This option enables or disables the implicit loading of symbols set in embedded code sections, which are not loaded explicitly with $offEmbeddedCode.

Example:

Set i(*);

$offECImplicitLoad
$onEmbeddedCode Python:
i = ['a','b']
print(i)
gams.set("i", i)
$offEmbeddedCode

This will result in a compilation error, since i is set in the embedded code block (by gams.set("i", i)), but not loaded with $offEmbeddedCode. If $onECImplicitLoad would be active, $offEmbeddedCode would be treated like $offEmbeddedCode i here.

See also ECImplicitLoad.

$[on][off]Embedded  ($offEmbedded)

Syntax:

$onEmbedded
$offEmbedded

This option enables or disables the use of embedded values in parameter and set data statements. If enabled, the explanatory text for set elements is concatenated with blank separators. For parameters, the embedded values get multiplied.

Example:

Set k / a,b /
   l / a /;
Set   i(k,l) / a.a 'aaaa cccc dddd'
      b.a 'bbbb cccc dddd' /;
Parameter m(k,l) / a.a 12
      b.a 24 /;
$onEmbedded
Set   j(k,l) / (a aaaa, b bbbb).(a cccc) dddd /;
Parameter n(k,l) / (a 1, b 2) .(a 3) 4 /;
Note that the explanatory text of the set elements in $i$ and $j$ as well as the values of the parameters $m$ and $n$ are identical.

$\text{on/offEmbeddedCode[S|V][.tag]}$

Syntax:

$\text{onEmbeddedCode[S|V][.tag]} \text{Connect|Python: [arguments]}
\begin{align*}
\text{Embedded code} \\
\{\text{Embedded code}\}
\end{align*}

$\text{offEmbeddedCode[.tag]} \{\text{symbol}<[=]\text{embSymbol[.dimX]}\}

This option is used to execute one or more lines of embedded code (e.g. Python code) while GAMS stays alive. The embedded code has access to GAMS symbols and can read and change them.

Note that parameter substitutions are permitted with $\text{onEmbeddedCode}$. The option $\text{onEmbeddedCode}$ has two more variants: $\text{onEmbeddedCodeS}$ and $\text{onEmbeddedCodeV}$. $\text{onEmbeddedCodeS}$ allows parameter substitutions like $\text{onEmbeddedCode}$, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended $S$. The option $\text{onEmbeddedCodeV}$ does not allow parameter substitutions but passes the code verbatim to the embedded code engine. The optional arguments given to $\text{onEmbeddedCode[S|V]}$ can be accessed in the Python code.

$\text{offEmbeddedCode}$ can be followed by a GAMS symbol or a list of GAMS symbols. If GAMS symbols are specified they get updated in the GAMS database after the embedded code got executed. The syntax $\text{symbol}<[=]\text{embSymbol[.dimX]}$ allows to load a one dimensional set from a symbol which was set in the embedded code that has even a higher dimensionality (here we call $<[=]$ the projection operator). GAMS tries to find the set symbol as a domain in the symbol embSymbol and uses the labels from this index position (with $<$ the first domain set from the right and with $<=$ from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix .dimX allows to pick a fixed index position ($X$ needs to be replaced by the desired index position).

One may add a tag to the $\text{onEmbeddedCode[S|V]}$ and $\text{offEmbeddedCode}$. For example, $\text{onEmbeddedCode.tagOne}$ would match with $\text{offEmbeddedCode.tagOne}$.

Example:

    country
city
mccCountry(cc,country<) Mapping between country and related elements in set cc
mccCity(cc,city<) Mapping between city and related elements in set cc;

$\text{onEmbeddedCode Python:}$
mccCountry = []
mccCity = []
for cc in gams.get("cc"):  
r = str.split(cc, " - ", 1)  
mccCountry.append((cc,r[0]))  
mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$\text{offEmbeddedCode mccCountry mccCity}$

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;
The data definition of the sets country and city happen while loading the set mccCountry and mccCity implicitly because of the implicit set definition in declarations of mccCountry and mccCity. Also note, that the list of output symbols does not need to be explicitly specified (for most embedded code engines). Performing the actual write in the embedded code (here via gams.set) is sufficient to signal GAMS to load the symbols. Providing an explicit list of symbols can be required (for embedded code engines without implicit loading) and be advantageous for code readability even for embedded code engines that support this. With the dollar control option $offECImplicitLoad providing an explicit list of output symbols can be enforced and setting symbols inside the embedded code not in the explicit list will lead to compilation errors.

The program above will result in the following listing file:

```
----  25 SET country
 Spain , USA , Germany, France
----  25 SET city
 Berlin, Bilbao, Cordoba, Madrid
 New York, Washington DC, Paris, Houston
 Munich, Lille, Seville, Bonn
Toulouse

----  25 SET mccCountry
 France - Paris .France
 France - Lille .France
 France - Toulouse .France
 Spain - Madrid .Spain
 Spain - Cordoba .Spain
 Spain - Seville .Spain
 Spain - Bilbao .Spain
 USA - Washington DC .USA
 USA - Houston .USA
 USA - New York .USA
 Germany - Berlin .Germany
 Germany - Munich .Germany
 Germany - Bonn .Germany

----  25 SET mccCity
 France - Paris .Paris
 France - Lille .Lille
 France - Toulouse .Toulouse
 Spain - Madrid .Madrid
 Spain - Cordoba .Cordoba
 Spain - Seville .Seville
 Spain - Bilbao .Bilbao
 USA - Washington DC .Washington DC
 USA - Houston .Houston
 USA - New York .New York
 Germany - Berlin .Berlin
 Germany - Munich .Munich
 Germany - Bonn .Bonn
```

Using the projection operator on the $offEmbeddedCode line the same task could be done like this:

```
Set cc / "France - Paris", "France - Lille", "France - Toulouse"
   "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
   "USA - Washington DC", "USA - Houston", "USA - New York",
```
"Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
country
city
mccCountry(cc,country)  Mapping between country and related elements in set cc
mccCity(cc,city)  Mapping between city and related elements in set cc;

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
r = str.split(cc, " - ", 1)
mccCountry.append((cc,r[0]))
mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode
country<mccCountry city<mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;

Please note that due to implicit output symbol loading, see \$offECImplicitLoad, the symbols
mccCountry and mccCity are not only used to define country and city but are also loaded
into GAMS because the embedded code uses gams.set to write to these symbols.

See also chapter Embedded Code Facility for more details.

$[on][off]Empty  ($onEmpty)

Syntax:

$onEmpty
$offEmpty

Setting $onEmpty allows empty data statements for list or table formats. Note that by default,
empty data statements will cause a compilation error.

Example:

Set    i   / 1,2,3 /;
Set    j(i) / /;
Parameter x(i)  "empty parameter" / /;
Table    y(i,i)  "headers only"
    1    2    3
;
$onEmpty
Set    k(i) / /;
Parameter xx(i)  "empty parameter" / /;
Table    yy(i,i)  "headers only"
    1    2    3
;

The resulting listing file will contain the following lines:
Set i / 1,2,3 /;
Set j(i) / /;
**** $460
Parameter x(i) "empty parameter" / /;
**** $460
Table y(i,i) "headers only"
 1 2 3
;
**** $462
Set k(i) / /;
Parameter xx(i) "empty parameter" / /;
Table yy(i,i) "headers only"
 1 2 3
;

Error Messages

460 Empty data statements not allowed. You may want to use $ON/OFFEMPTY
462 The row section in the previous table is missing

Empty data statements are most likely to occur when data is being entered into the GAMS model by an external program. This problem may be overcome with the option $onEmpty.

Note

The empty data statement may only be used with symbols which have a known dimension. If the dimension is also derived from the data, the option $phantom should be used to generate 'phantom' set elements.

The option $onEmpty in conjunction with the option $onMulti and the save and restart feature may be used to set up a model and add data later.

$[on][off]End   ($offEnd)

Syntax:

$onEnd
$offEnd

This option offers an alternative syntax for flow control statements. The option $onEnd causes the following words to be regarded as keywords: do, endLoop, endIf, endFor and endWhile. They are used to close the language constructs loop, if, for and while respectively.

Example:

Note

The standard syntax is given as an end-of-line comment.

Set i / 1*3 /;
Scalar cond / 0 /;
Parameter a(i) / 1 1.23, 2 2.65, 3 1.34/;
$solCom //
$onEnd

loop i do
  // loop (i,
display a;
  // display a;
endLoop;

if (cond) then
    display a;
else
    a(i) = a(i)/2;
    display a;
endIf;

for cond = 1 to 5 do
    a(i) = 2 * a(i);
endFor;

while cond > 3 do
    a(i) = a(i) / 2;
    cond = cond-1;
endWhile;

Observe that the alternative syntax is more in line with the syntax used in some of the popular programming languages.

Attention

Setting the option $onEnd will make the alternative syntax valid, and at the same time it will make the standard syntax invalid. Therefore the two forms of the syntax will never be valid simultaneously.

$[on][off]EolCom ($offEolCom)

Syntax:

$_onEolCom$
$_offEolCom$

This option acts as a switch to control the use of end-of-line comments. Note that by default, the end-of-line comment symbol is set to !! but the processing is disabled.

Example:

$_onEolCom$
Set i /1*2/ ; !! set declaration
Parameter a(i) ; !! parameter declaration

Observe that after the option $_onEolCom$ has been specified, comments may be entered on the same line as GAMS code.

Note

The option $_eolCom$ automatically sets $_onEolCom$.

See also section Comments.

$[on][off]Eps ($offEps)

Syntax:
$onEps
$offEps

This option is used to interpret zero values as EPS when non-scalar parameters, variable attributes, or equation attributes are defined or loaded (e.g. from GDX or embedded code) at compile time. This can for example be useful if the value of zero is overloaded with existence interpolation.

See option zeroToEps for the execution-time equivalent.

Example 1:

Consider the following example where zero values are defined for a one dimensional parameter, once under $onEps and $offEps.

Set i / one, two, three, four /;
Parameter a(i) /
$onEps
  one 0
$offEps
  two 0
  three EPS /;
Display a ;

The outcome generated by the display statement follows:

---- 8 PARAMETER a
one     EPS, three EPS

Note that only those entries specifically entered as 0 are treated as EPS.

Example 2:

Consider the following example where zero values are loaded from a GDX file, once under $offEps (default) and $onEps (derived from test library model [ZEROTOEPS1]).

Set s,t;
Parameter p(s<,t<)
  pOnEps(s,t);
$gdxIn zeropar.gdx
$onText
P looks like this:
  t1   t2   t3
s1 0  0.843267
s2 0.224053 0.349831 0
s3 0  5
$offText
$load p
$onEps
$load pOnEps=p
display p, pOnEps;

The outcome generated by the display statement follows:
--- 15 PARAMETER p

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0.843</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>0.224</td>
<td>0.350</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>5.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- 15 PARAMETER pOnEps

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>EPS</td>
<td>0.843</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>0.224</td>
<td>0.350</td>
<td>EPS</td>
</tr>
<tr>
<td>s3</td>
<td>EPS</td>
<td>5.000</td>
<td></td>
</tr>
</tbody>
</table>

Example 3:

Consider the following example where zero values are loaded from embedded Python code, once under $offEps (default) and $onEps (derived from test library model [ZERO TO EPS1]).

Set s / s1*s3 /;
Parameter pEC(s)
  pECOnEps(s);
$onEmbeddedCode Python:
p = [(s1',1), (s3',0)]
gams.set('pEC',p)
$offEmbeddedCode pEC
$onEps
$onEmbeddedCode Python:
p = [(s1',1), (s3',0)]
gams.set('pECOnEps',p)
$offEmbeddedCode pECOnEps
display pEC, pECOnEps;

The outcome generated by the display statement follows:

--- 14 PARAMETER pEC

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
s1 1.000 |

--- 14 PARAMETER pECOnEps

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
s1 1.000, s3 EPS |

$[on][off]EpsToZero  ($offEpsToZero)

Syntax:

$onEpsToZero
$offEpsToZero
This option is used to write EPS values as zero when parameters, or variable and equation levels get written to GDX at compile time.

See option EpsToZero for the execution-time equivalent.

**Example:**

Usually GAMS squeezes out 0 values when writing to GDX. To get them written anyway, one can define them as EPS (e.g. by using $onEps) and set $onEpsToZero:

```plaintext
Set i / zero, one, two /;
$onEps
Parameter a(i) / zero 0
      one 1
      two 2 /;

$onEpsToZero
.gdxUnload 'zero.gdx' a
```

$[on][off]Expand ($offExpand)

**Syntax:**

$onExpand
$offExpand

This option changes the processing of macros that appear in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch $onExpand enables the recognition and expansion of macros in the macro argument list and $offExpand will restore the default behavior.

**Example:**

```plaintext
variable x(*,*);
$macro f(i) sum(q, x(i,q))
$macro equ(x) equation equ_&x; equ_&x.. &x =e= 0;
equ(f(i))
```

The macro expansion of the code above will result in an equation definition that reads as follows:

```plaintext
equation equ_f(I); equ_f(i).. f(i) =e= 0;
```

If we compile the code under $onExpand the argument f(i) is expanded before the macro equ() gets expanding resulting in the following (incorrect) code:

```plaintext
equation equ_sum(q, x(i,q)); equ_sum(q, x(i,q)).. sum(q, x(i,q)) =e= 0;
```

For further information, see section **Macros in GAMS** below.

$[on][off]ExternalInput ($offExternalInput)

**Syntax:**

$onExternalInput
$offExternalInput

$onExternalInput and $offExternalInput mark the beginning and end of a well defined section in which data is loaded implicitly from an external data source, namely from a GDX file specified by the command line parameter IDCGDXInput.
Note

- If none of the command line parameters `IDCGDXInput`, `IDCGenerateGDXInput`, `IDCJSON`, `IDCGenerateJSON`, or `IDCGenerateGDX` is set, nothing - but a few checks for consistency - will happen by setting `$onExternalInput` and `$offExternalInput`.
- If at least one of the before mentioned command line parameters is set, an additional license check will be triggered: Without the MIRO Connector License not more than 5 indexed symbols can be declared as external input or external output.

For symbols, which are declared, while `$onExternalInput` is active, data in data statements of the declaration is ignored. Instead, the data is loaded from the GDX file specified by `IDCGDXInput`. Also for those symbols, if they are referenced with `$load` or `$offEmbeddedCode` while `$onExternalInput` is active, the data is loaded from the GDX file specified by `IDCGDXInput` instead of the defined data source and symbol renaming as well as the projection operator are ignored in this case.

In addition, the following dollar control options are ignored, while `$onExternalInput` is active and `IDCGDXInput` is set:

- `$batInclude`
- `$call`
- `$call.Async[NC]`
- `$call.checkErrorLevel`
- `$gdxIn`
- `$hiddenCall`
- `$hiddenCall.Async[NC]`
- `$hiddenCall.checkErrorLevel`
- `$include`
- `$libInclude`
- `$onEmbeddedCode[S|V]` (including the following embedded code until `$offEmbeddedCode` is hit)
- `$sysInclude`

Example:

Let's look at a modified version of the simple example presented here:

Set
  i 'canning plants'
  j 'markets';

$onExternalInput
$call mdb2gms l=Sample.accdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i<j) 'distance in thousands of miles' /
$include distances.inc
/;
$offExternalInput

display d;

Without setting a command line parameter, this runs in the same way as described in the other chapter, so by default, setting `$onExternalInput` and `$offExternalInput` does not make much of a difference. Now, let's run it with `IDCGDXInput` being set to `transsol`, which is a GDX file generated by running the transportation model `[TRNSPORT]`. There will be some new messages in the log:
First, we notice, that the file *transsol.gdx* will be opened for input. This happens implicitly with `$onExternalInput`. Then, both the `$call` as well as the `$include` lines were ignored, since we do not want to query the data source explicitly defined in the model, but the GDX file defined by IDCGDXInput. Note, that the data statement in the declaration of $d$ is actually empty when we ignore the `$include` statement. While `$onExternalInput` this does not lead to an error, but behaves as if `$onEmpty` was set. Looking at the output of the `display` statement in the lst file, one can see that the expected data was loaded:

```
----  13 PARAMETER d distance in thousands of miles

new-york  2.500  1.700  1.800
seattle    2.500  1.700  1.800
san-diego  2.500  1.800  1.400
```

If any of the options IDCGDXInput, IDCGDXOutput, IDCJSON or their Generate versions, e.g. IDCGenerateGDXInput, has been set, a MIRO connector license is required. See also `$[on][off]ExternalOutput`.

### `$[on][off]ExternalOutput` ($offExternalOutput)

**Syntax:**

```
$onExternalOutput
$offExternalOutput
```

$onExternalOutput and $offExternalOutput mark the beginning and end of a well defined section in which data is written implicitly to an external data file, namely to a GDX file specified by the command line parameter IDCGDXOutput or IDCGenerateGDXOutput.

**Note**

- If none of the command line parameters IDCGDXOutput, IDCGenerateGDXOutput, IDCJSON, IDCGenerateJSON, or IDCGenerateGDX is set, nothing - but a few checks for consistency - will happen by setting $onExternalOutput and $offExternalOutput.
- If at least one of the before mentioned command line parameters is set, an additional license check will be triggered: Without the MIRO Connector License not more than 5 indexed symbols can be declared as external output or external input.

**Example:**

This is a slightly modified code snippet of the transportation model [TRNSPORT]:

```
...$onExternalOutput
Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;
Variable
   x(i,j) 'shipment quantities in cases'
```
$offExternalOutput
z 'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
$onExternalOutput
cost 'define objective function'
$offExternalOutput
supply(i) 'observe supply limit at plant i'
demand(j) 'satisfy demand at market j';

Without setting a command line parameter, this runs in the same way as the original model, so by default, setting $onExternalOutput and $offExternalOutput does not make much of a difference. Now, let's run it with IDCGDXOutput being set to transOut.gdx. This generates the file transOut.gdx with the following content:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Name</th>
<th>Type</th>
<th>Dim</th>
<th>Records</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>transport cost in ...</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>Variable</td>
<td>2</td>
<td>5</td>
<td>shipment quantity</td>
</tr>
</tbody>
</table>

So, this implicit generation of the output file is basically equivalent to the following explicit statement:

execute_unload 'transOut.gdx', c, x, cost;

If any of the options IDCGDXInput, IDCGDXOutput, IDCJSON or their Generate versions, e.g. IDCGenerateGDXInput, has been set, a MIRO connector license is required.

See also $[on][off]ExternalInput.

$[on][off]Filtered  ($onFiltered)

Syntax:

$onFiltered
$offFiltered

This dollar control option controls how GAMS loads data from GDX with gdxLoad, $load, $loadR, and $loadM. With $onFiltered the $load* calls behave like $loadFiltered* calls. With $offFiltered the $load* calls behave like $loadDC* calls. This dollar control option also controls how GAMS loads symbols on the $offEmbeddedCode line. With $onFiltered data is read filtered while $offFiltered triggers a domain check of the data. The gams.set Python function allows to overwrite the setting of this dollar control option with the argument domCheck=DomainCheckType.CHECKED or domCheck=DomainCheckType.FILTERED.

$[on][off]Global  ($offGlobal)

Syntax:
When an include file is inserted, it inherits the dollar control options from the higher level file. However, the dollar control option settings specified in the include file do not affect the higher level file. This convention is common among most scripting languages or command processing shells. In some cases, it may be desirable to break this convention. This option allows an include file to change the options of the parent file as well.

Example:

```plaintext
$include 'inc.inc'
$hidden after first call to include file
$onGlobal
$include 'inc.inc'
$hidden after second call to include file
```

The file `inc.inc` contains the following lines:

```plaintext
$onDollar
$hidden text inside include file
```

The the echo print of the resulting listing file follows:

```
INCLUDE D:\GAMS\INC.INC
2 $onDollar
3 $hidden text inside include file
#include D:\GAMS\INC.INC
7 $onDollar
8 $hidden text inside include file
9 $hidden after second call to include file
```

Note that the dollar control option `$onDollar` inside the include file does not affect the parent file until `$onGlobal` is set. The text following the option `$hidden` is then echoed to the listing file.

$[on][off]IDCProtect  ($onIDCProtect)

Syntax:

```plaintext
$onIDCProtect
$offIDCProtect
```

This controls if symbols which are declared as external input can be changed at execution time. With `$onIDCProtect` external input symbols are protected from being changed; `$offIDCProtect` allows it.

The initial state of this can be set with the command line parameter `IDCProtect`.

$[on][off]Include  ($onInclude)

Syntax:

```plaintext
$onInclude
$offInclude
```
This option controls the listing of the expanded include file name in the listing file.

**Example:**

```
$include 'inc.inc'
$offInclude
$include 'inc.inc'
```

We assume that the file `inc.inc` contains the following lines:

```
$onDollar
$hidden Text inside include file
```

The resulting listing file will contain the following lines:

```
INCLUDE C:\tmp\inc.inc
2 $onDollar
3 $hidden Text inside include file
6 $onDollar
7 $hidden Text inside include file
```

Note that the include file name is echoed the first time the include file is used. However, the include file name is not echoed after `$offInclude` has been set. Also, the include file summary and the creation of an **Expanded Include File** is suppressed if `$offInclude` is set.

**$[on][off]Inline**  (**$offInline**)  

**Syntax:**

```
$onInline
$offInline
```

This option acts as switch to control the use of in-line comments. Note that by default, the in-line comment symbols are set to the two character pairs `/*` and `*/` but the processing is disabled. In-line comments may span several lines till the end-of-comment characters are encountered.

**Example:**

```
$onInline
Set i /* The default comment symbols are now
active. These comments can continue
  to additional lines till the closing
  comments are found. */ /i*i3 /;
```

Note

- The option `$inlineCom` automatically sets `$onInline`.
- *Nested* in-line comments are illegal unless the option `$onNestCom` is set.

See also section **Comments**.

**$[on][off]Listing**  (**$onListing**)  

**Syntax:**

```
```
This option controls the echoing of input lines to the compilation output of the listing file. Note that suppressed input lines do not generate entries in the symbol and reference sections that appear at the end of the compilation output. Lines with errors will always be listed.

Example:

```plaintext
Set i /0234*0237/
    j /a,b,c/ ;
Table x(i,j) "very long table"
    a b c
    0234 1 2 3
$offListing
    0235 4 5 6
    0236 5 6 7
$onListing
    0237 1 1 1
; 
```

The resulting listing file will contain the following lines:

```plaintext
1 Set i /0234*0237/
2    j /a,b,c/ ;
3 Table x(i,j) very long table
4    a b c
5 0234 1 2 3
10 0237 1 1 1
```

Note that the lines in the source file between the options `$offListing` and `$onListing` are not echoed to the listing file.

Note

For some projects the listing file can become huge and can take significant time to be written. This time can be saved by setting `$offListing` at the beginning of the input file and `$onListing` just before the parts one is interested in, or not at all, if one does not look at the listing file anyway.

$[on][off]Local ($onLocal)

Syntax:

$onLocal
$offLocal

The suffix `.local` attached to the name of a controlling set will use an *implicit alias* within the scope of the indexed operation or on the left-hand side of an assignment statement. This feature is particularly useful in the context of nested macros.

Example:

```plaintext
Set i /1*3/; alias(i,j);
Parameter xxx(i,j) / 1.1 1, 2.2 2, 3.3 3, 1.3 13, 3.1 31 /;
display xxx;

Parameter p(i);
p(i.local) = sum(j, xxx(i,j));
display p;
```
Note that in the assignment statement the set \( i \) on the right-hand side is controlled by \( i\.local \) on the left-hand side. Thus we have the following values for the two parameters:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 1.000 & 13.000 \\
2 & 2.000 & \\
3 & 31.000 & 3.000 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 1.000 & 13.000 \\
2 & 2.000 & 3.000 \\
\end{array}
\]

In the example above, the suffix \( .local \) appeared one time on the left-hand side. The option \$onLocal \ allows the suffix \( .local \) to appear more than one time attached to the same symbol. Consider the following example that extends the example above:

Parameter \( g(i,i) \);
\( g(i\.local-1,i\.local) = xxx(i,i) \);
display \( g \);

Note that in the assignment statement of \( g \) the suffix \( .local \) attached to the set \( i \) appears two times on the left-hand side. The question arises whether the reference to the set \( i \) on the right-hand side refers to the first or the second instance of \( .local \) on the left-hand side. The assignment statement may alternatively be written in the following way using an explicit alias statement:

\[
\text{alias } (i,i1,i2);
\]
\( g(i1-1,i2) = xxx(i2,i2) \);

display \( g \);

Thus it becomes clear that the symbol on the right-hand side refers to the controlling index that enters last (here the second one). The output generated by the display statement follows:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 1.000 & 2.000 & 3.000 \\
2 & 1.000 & 2.000 & 3.000 \\
\end{array}
\]

Observe that the multiple use of the suffix \( .local \) on the same symbol is considered an error with the option \$offLocal \.

Note that it is also allowed to combine the original index with an index suffixed with \( .local \). Consider the following alternative formulation:

\( g(i\.local-1,i) = xxx(i,i) \);

Note that in this case the index suffixed with \( .local \) takes precedence and the reference of \( i \) on the right-hand side refers to the index \( i\.local \) even though \( i \) is entered last. Observe that this statement even works with \$offLocal \ as the suffix \( .local \) appears only once.

See also section Macros in GAMS below.

\$[on][off]Log \ ($onLog$)

Syntax:

\$onLog
\$offLog
This option acts as a switch that controls logging information about the line number and memory consumption during compilation. This is scoped like the option $on/offListing applying only to included files and any subsequent included files but reverting to the setting $onLog in the parent files (if it was not changed there as well).

Example:

```
Set i /i1*i20000000/;
$include inc.inc
Set l /l1*l12000000000/;
```

The file inc.inc looks like this:

```
Set j /j1*j20000000000/;
$offLog
Set k /k1*k20000000000/;
```

The generated log will contain the following lines:

```
--- test.gms(1) 1602 Mb 5 secs
--- test.gms(2) 1602 Mb
--- inc.inc(1) 3122 Mb 6 secs
--- test.gms(3) 6161 Mb 14 secs
```

Note that the first line of both the parent and the include file got logged, but not the third line of the include file, after $offLog was set. The last line of the parent file got logged again.

$[on][off]Macro ($onMacro)

**Syntax:**

```
$onMacro
$offMacro
```

Enables or disables the expansion of macros defined by $macro.

**Example:**

```
$macro oneoverit(y) 1/y
$offMacro
y = oneoverit(x1);
display y;
```

causes an error because the macro oneoverit in line 3 can not be expanded.

$[on][off]Margin ($offMargin)

**Syntax:**

```
$onMargin
$offMargin
```

This option controls margin marking, that means if margins set by the options $minCol and $maxCol, should be marked in the lst file.

**Example:**
Now we have set i "plant" / US, UK /; This defines i
turned on the scalar x / 3.145 /; A scalar example.
margin marking. parameter a, b;
Define some
parameters.

The lst file will contain this:

2  Now we have | Set   i "plant" / US, UK /; | This defines i
3  turned on the | Scalar x / 3.145 /; | A scalar example.
4  margin marking. | Parameter a, b; | Define some
5                                   | parameters.

Note that any statements between columns 1 and 19 and any input beyond column 52 are
treated as comments. These margins are marked with | on the left and right.

See also section Comments.

$[on][off]Multi ($offMulti)

Syntax:

$onMulti
$offMulti

This option controls multiple data statements or tables. By default, GAMS does not allow
data statements to be redefined. If this option is activated the second or subsequent data
statements are merged with entries of the previous ones. Note that all multiple data statements
are performed before any other statement is executed.

Note

• There is also $onMultiR, which behaves similarly, but replaces existing data instead
  of merging into it. Compare the example here and there to see the difference.
• When $onMulti is active $load behaves like a $loadM.
• The initial state of this dollar control option can be set through the command line
  parameter multi.

Example:

Consider the following slice of code. The list after the end of line comment describes the
complete content of the symbol x after the data statement has been processed:

$eolCom //
set i / i1*i10 /;
parameter x(i) / i1*i3 1 / // /i1 1,i2 1,i3 1/
$onMulti
parameter x(i) / i7*i9 2 / // /i1 1,i2 1,i3 1,i7 2,i8 2,i9 2/
parameter x(i) / i2*i6 3 / // /i1 1,i2 3,13 3,14 3,15 3,16 3,17 2,i8 2,i9 2/
parameter x(i) / i3*i5 0 / // /i1 1,i2 3,16 3,17 2,i8 2,i9 2/
$offMulti
display x;

Note that the repeated parameter statements would have resulted in a compilation error
without the presence of the option $onMulti. The result of the display statement in the listing
file follows:
4.41 Dollar Control Options

---

8 PARAMETER x

1 1.000, 2 3.000, 6 3.000, 7 2.000, 8 2.000, 9 2.000

Note that \textit{x(\textquoteleft{i1\textquoteleft})} is assigned the value of 1 with the first data and is not affected by any of the subsequent data statements. \textit{x(\textquoteleft{i3\textquoteleft})} on the other hand is reset to 3 by the third data statement and wiped out with 0 in the fourth data statement.

\textbf{Attention}

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option \texttt{$onMulti} since it allows data initializations to be performed more than once. See section GAMS Compile Time and Execution Time Phase for details.

Consider the following example:

Scalar a /12/;
a=a+1;
\texttt{$onMulti}
Scalar a /20/;
display a;

Note that the two \texttt{scalar} data initialization statements and the option \texttt{$onMulti} are processed before the assignment statement \texttt{a=a+1}. As a result, the final value of \texttt{a} will be 21. The output of the display statement follows:

---

5 PARAMETER a = 21.000

Observe that the option \texttt{$onEmpty} in conjunction with the option \texttt{$onMulti} and the save and restart feature may be used to set up a model and add data later. See example in section Advanced Separation of Model and Data for details.

\texttt{$onMultiR/onMulti (\texttt{$offMulti})}

\textbf{Syntax:}

\texttt{$onMultiR}
\texttt{$offMulti}

This option controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is activated the second or subsequent data statements replace the previous ones. Note that all multiple data statements are performed before any other statement is executed.

\textbf{Note}

- There is also \texttt{$onMulti}, which behaves similarly, but \textit{merges} into existing data instead of replacing it. Compare the example here and there to see the difference.
- In contrast to \texttt{$onMulti}, \texttt{$onMultiR} also allows to redefine an equation and a macro.
- The initial state of this dollar control option can be set through the command line parameter \texttt{multi}.

\textbf{Example:}

Consider the following slice of code. The list after the end of line comment describes the complete content of the symbol \textit{x} after the data statement has been processed:
Note that the repeated parameter statements would have resulted in a compilation error without the presence of the option $onMultiR. The result of the display statement in the listing file follows:

```
---- 9 PARAMETER x
      ( ALL       0.000 )
```

Note

- When using a redefinition with $onMultiR to remove elements from a set, which was used as domain of another symbol, also the dependent symbol gets reduced. This can happen through data statements, but also with $clear and $load/$loadDC.
- When $onMultiR is active $load behaves like a $loadR.

Example:

```
Set i /i1*i5/;
Parameter p(i) /#i 3/;
$onMultiR
Set i /i3*i6/;
Display p;
```

Looking at the output from the Display statement one can see, that the entries for i1 and i2 were removed by the second definition of i:

```
---- 7 PARAMETER p
i3 3.000,  i4 3.000,  i5 3.000
```

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option $onMultiR since it allows data initializations to be performed more than once. See section GAMS Compile Time and Execution Time Phase for details.

Consider the following example:

```
Scalar a /12/;
a=a+1;
$onMultiR
Scalar a /20/;
Display a;
```
Note that the two scalar data initialization statements and the option $onMultiR are processed before the assignment statement a=a+1. As a result, the final value of a will be 21. The output of the display statement follows:

```plaintext
---- 5 PARAMETER a = 21.000
```

Observe that the option $onEmpty in conjunction with the option $onMultiR and the save and restart feature may be used to set up a model and add data later. See example in section Advanced Separation of Model and Data for details.

### $[on][off]$NestCom  ($offNestCom$)

**Syntax:**

```plaintext
$onNestCom
$offNestCom
```

This option controls nested in-line comments. It makes sure that the open-comment and close-comment characters match.

**Example:**

```plaintext
$inlineCom { } onNestCom
{ nesting is now possible in comments { braces have to match } }
```

See also $inlineCom, $onInLine and section Comments.

### $[on][off]$Order  ($onOrder$)

**Syntax:**

```plaintext
$onOrder
$offOrder
```

Lag and lead operations and the ord function require the referenced set to be ordered and constant. In some special cases users might want to use those operations on dynamic and/or unordered sets. The option $on/offOrder has been added to locally relax the default requirements. The use of this option comes with a price, the system will not be able to diagnose odd and incorrect formulations and data sets.

**Example:**

```plaintext
Parameter p(t2);
$offOrder
p(t2) = ord(t2);
display t2,p;
```

Without the $offOrder the compilation of the line p(t2) = ord(t2); would have triggered a compilation error. The ordinal numbers assigned here are probably not what one expects. The element 1987 gets ordinal number 1 although it seems to be last last in the set. The ordinal numbers are assigned in the order the set is stored internally in GAMS. This order is also used when displaying the set t2:
---- 6 SET t2
---- 6 PARAMETER p
1987 1.000, 1983 2.000, 1984 3.000, 1985 4.000, 1986 5.000

$[on][off]Put[S][V][.tag]

Syntax:

File myputfile;
put myputfile;
$onPut[S|V][.tag]
{text}
$offPut[.tag]

The pair $onPut - $offPut causes a block of text to be placed in a put file at run-time. The is one of the few dollar control options that operate at run time. The $ in the first column usually indicates action at compile time.

Note that parameter substitutions are not permitted with $onPut. The option $onPut has two more variants: $onPutS and $onPutV. $onPutS allows parameter substitutions while the option $onPutV does not allow parameter substitutions, like $onPut, so it is just a synonym which makes it more obvious that the text is written verbatim with the appended V.

Example:

$set it TEST
File myputfile;
put myputfile;
$onPutS
Line 1 of text "%it%"
Line 2 of text %it%
$offPut

This code generates the put file myputfile.put with the following content:

Line 1 of text "TEST"
Line 2 of text TEST

Note that the compile-time variable %it% was replaced by TEST. However, if the option $onPutV is used instead, then %it% will not be substituted:

$set it TEST
File myputfile
put myputfile
$onPutV
Line 1 of text "%it%"
Line 2 of text %it%
$offPut

The resulting file myputfile.put will contain the following lines:

Line 1 of text "%it%"
Line 2 of text %it%
One may add a tag to the $onPut[S][V]$ and $offPut$. This allows to use nested put blocks like this:

```gams
File myputfile / outer.gms /;
put myputfile;
$onPut.outer
File myputfile;
put myputfile;
$onPut.inner
Line 1 of text
Line 2 of text
$offPut.inner
$offPut.outer
```

The resulting file `outer.gms` looks like this:

```gams
File myputfile;
put myputfile;
$onPut.inner
Line 1 of text
Line 2 of text
$offPut.inner
```

`$[on]offUpper ($offUpper)`

**Syntax:**

```gams
$onUpper
$offUpper
```

GAMS code echoed to the listing file is written in upper case after `$onUpper`. The default of mixed code echoing is restored with `$offUpper`.

**Note**

The default case of the echo print can be set using the command line option `case`.

**Example:**

```gams
Set
  i 'Canning plants' / seattle, san-diego /
  j 'Markets' / new-york, chicago, topeka /;
$onupper
Parameter
  a(i) 'Capacity of plant i in cases'
  / seattle 350
  san-diego 600 /
$offupper
  b(j) 'Demand at market j in cases'
  / new-york 325
  chicago 300
  topeka 275 /;
```

The generated listing file looks like this:
Set
  i 'Canning plants' / seattle, san-diego /
  j 'Markets' / new-york, chicago, topeka /;

PARAMETER
  A(I) 'CAPACITY OF PLANT I IN CASES'
    / SEATTLE 350
    SAN-DIEGO 600 /

  b(j) 'Demand at market j in cases'
    / new-york 325
    chicago 300
    topeka 275 /;

$[on][off]Recurse ($offRecurse)

Syntax:

$onRecurse
$offRecurse

This option controls whether it is permitted for a file to include itself.

Example:

The following GAMS program result in a recursive inclusion of the program itself:

$onRecurse
$include "%gams.input%"

Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

In the following example that prints a string and then the reverse string the nesting level is less that 40 and one get some kind of recursion at compile time:

$onEchoV > reverse.gms
$ifthene %1=%3+1
  put ' ';
$ exit
$endif
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$eval posPlus1 %1+1
$batInclude reverse %posPlus1% %2 %3
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$offEcho

set chars / A*Z /, code / 65*90 /, map(chars,code) / #chars:#code /;
file fx /''/; put fx;
$onRecurse
$batInclude reverse 1 RACECAR 7
put /;
$batInclude reverse 1 LAGER 5

The log will print the following lines:
4.41 Dollar Control Options

--- Starting execution: elapsed 0:00:00.067
RACECAR RACECAR
LAGER REGAL
*** Status: Normal completion

$[on][off]StrictSingleton ($onStrictSingleton)

Syntax:

$onStrictSingleton
$offStrictSingleton

If the option $onStrictSingleton is active, a compilation error is triggered if a data statement for a singleton set contains more than one element. After activating the option $offStrictSingleton GAMS will take the first element of a singleton set that was declared with multiple elements as the valid element, the other elements are disregarded and there is no error. The option to control this behavior at runtime is strictSingleton.

Example:

The first element is not always the one that appears in the data statement first as the following example shows:

```gams
set i /1,2/
$offStrictSingleton
singleton set ii(i) /2,1/;
display ii;
```

The set ii contains the element 1 because it is the first in the GAMS label order as the display statements shows:

```gams
---- 4 SET ii

1
```

$[on][off]SuffixDLVars  ($onSuffixDLVars)

Syntax:

$onSuffixDLVars
$offSuffixDLVars

This option controls if it is allowed to use suffixes on variables with limited domains in a model. By default this is not allowed but this can be changed by setting $onSuffixDLVars.

Example:

```gams
Set n / n1*n100 /
   t / t1*t10 /
   sub(n,t) / #n:#t /;
Variable z, x(n,t);
Equation obj;

x.up(n,t) = uniformInt(7,13);
obj.. z =e= sum((n,t),x(n,t)/x.up(n,t));
Model m / all, x(sub) /;
Solve m min z use lp;
```
This causes the following error by default:

*** Problem in Model Generation: Variable "x" appears with suffix "Up".
*** Error at line 12: Cannot use domain limited variables with suffix in model - use $onSuffixDLVars

The purpose is to make the user aware of the fact, the she uses x.l, which would also be affected by the domain restriction of x(sub) in the model statement. So, if the user made sure, that this is all intended, setting $onSuffixDLVars before the solve statements overcomes this error. Though, with the example given here, this would actually lead to a "division by zero" error, since some x.up were excluded leaving a 0 as divisor.

See also related command line parameter SuffixDLVars.

$[on][off]SymList  ($offSymList)

Syntax:

$onSymList
$offSymList

This option controls whether the symbol listing map appears in the compilation output of the listing file. The symbol listing map contains the complete listing of all symbols that have been defined and their explanatory text. The entries are in alphabetical order and grouped by symbol type.

Example:

The symbol listing map generated by running [TRNSPORT] with $onSymList is as follows:

Symbol Listing

SETS

i   canning plants
j   markets

PARAMETERS

a   capacity of plant i in cases
b   demand at market j in cases
c   transport cost in thousands of dollars per case
d   distance in thousands of miles
f   freight in dollars per case per thousand miles

VARIABLES

x   shipment quantities in cases
z   total transportation costs in thousands of dollars

EQUATIONS

cost   define objective function
demand satisfy demand at market j
supply observe supply limit at plant i

MODELS

transport
This serves as a simple description of the symbols used in a model and may be used in reports and other documentation. For further information, see section The Symbol Listing Map.

$[on][off]$SymXRef  ($offSymXRef$)

Syntax:

$onSymXRef
$offSymXRef

This option controls the following:

- Collection of cross references for symbols like sets, parameters, variables, acronyms, equations, models and put files.
- Symbol cross reference report of all collected symbols in the compilation output of the listing file. For details, see section The Symbol Reference Map.
- Listing of all referenced symbols and their explanatory text by symbol type in listing file. This listing may also be activated with the option $onSymList$.

Example:

$onSymXRef
Set i / 1*6 /, k;
$offSymXRef
Set j(i) "will not show" / 1*3 /;
$onSymXRef
k('1') = yes;

The resulting listing file will contain the following symbol reference map and symbol listing map:

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TYPE</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>SET</td>
<td>declared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>k</td>
<td>SET</td>
<td>declared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 assigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

SETS

i
k

Note that the set j does not appear in these listings because the listing was deactivated with the option $offSymXRef$ in line 3 of the code above.

$[on][off]$Text

Syntax:

$onText
$offText

The pair $onText - $offText encloses comment lines. Line numbers in the compiler listing are suppressed to mark skipped lines.

Example:
* Standard comment line
$onText
Everything here is a comment until we encounter the closing $offText like the one below
$offText
* Another standard comment line

The echo print of the resulting listing file will contain the following lines:

```
1  * Standard comment line
   Everything here is a comment
   until we encounter the closing $offText
   like the one below
7  * Another standard comment line
```

Attention
GAMS requires that every $onText has a matching $offText and vice versa.

See also section Comments.

$[on][off]UElList ($offUElList)

Syntax:

```
$onUElList
$offUElList
```

This option controls the complete listing of all set elements that have been entered in the compilation output of the listing file. For details see section The Unique Element Listing Map.

Example:

The unique element listing in the listing file generated by running the model [TRNSPORT] with $onUElList follows:

Unique Element Listing

Unique Elements in Entry Order

1 seattle san-diego new-york chicago topeka

Unique Elements in Sorted Order

1 chicago new-york san-diego seattle topeka

Note that the sorted order is not the same as the entry order. For more information, see section Ordered and Unordered Sets.

$[on][off]UElXRef ($offUElXRef)

Syntax:

```
$onUElXRef
$offUElXRef
```

This option controls the collection and listing of cross references of set elements in the compilation output. For more information, see section The Unique Element Listing Map.

Example:

Set i "set declaration" / one, two, three /, k(i);
$onUElXRef
k('one') = yes;
$offUElXRef
k('two') = yes;
$onUElXRef
k('three') = yes;

The resulting listing file will contain the following unique element reference report:

Unique Element Listing

ELEMENT REFERENCES

one index 3
three index 7

Note that the element two does not appear in this listing because the listing was deactivated with the option $offUElXRef in line 4 of the code above.

$[on][off]UNDF  ($offUNDF)

Syntax:

$onUNDF
$offUNDF

This option controls the use of the special value UNDF which indicates a result is undefined. For details see section Extended Range Arithmetic. By default, UNDF is not permitted to be used in assignments. This may be changed with the option $onUNDF.

Example:

Scalar x;
$onUNDF
x = UNDF;
Display x;

The output of the display statement follows:

---- 4 PARAMETER x = UNDF

Note that an error would have been triggered without the use of $onUNDF. The option $offUNDF will return the system to the default, where UNDF may not be used in assignments.

$[on][off]Uni  ($offUni)

Syntax:

$onUni
$offUni
This controls whether the compiler checks the referential integrity (see section Domain Checking) of the code. This is an essential part of good GAMS programming and it is highly recommend to declare symbols with proper domains. With the universe as a domain the compiler does not help the user with easy-to-make mistakes, like swapping indexes, \texttt{a(i,j)} versus \texttt{a(j,i)}. By default something like this would generate an error, if \texttt{a} was declared as \texttt{a(i,j)}. Such an error could be ignored, by setting \$onUni, which can be useful in few situations, when accessing a symbol with a set that is not the domain or a subset of the domain. For example, we could read data of a union of sets that already exist. We could use the universe as the domain for that symbol, but perhaps we need to protect the referential integrity of this symbol too.

**Example:**

```gams
Set fruit / apple, pear /
    veggie / carrot, pea /
    produce / #fruit, #veggie /;
Parameter produceCalories(produce) "per 100g" / apple 52, pear 57, carrot 41, pea 81 /
    fc(fruit) "calories per 100g"
    vc(veggie) "calories per 100g";
$onUni
fc(fruit) = produceCalories(fruit);
vc(veggie) = produceCalories(veggie);
$offUni
display fc, vc;
```

So when assigning \texttt{fc} we only access \texttt{produceCalories} with \texttt{fruit}. We could reverse the order of declaration of \texttt{fruit, veggie} and \texttt{produce} and use a proper subdomain, but sometimes data flow and input don't allow that.

**Attention**

When the GAMS compiler operates under \$onUni it treats all symbols as being declared over the universe. So all domain checking is gone. We can set elements in a symbols that normally can't be entered. This can also lead to strange effects:

```gams
set i / 1*2 /
    j / a,b /;
parameter pi(i);
$onuni
pi(j) = 1;
$offuni
* We will see elements from j in pi
display pi;
* The following should only clear the i-elements from pi, but it clears the
* entire symbol, because GAMS knows it's doing this to the entire domain and
* takes a shortcut.
p(i) = no;
display pi;
```

**$[on][off]Verbatim**

**Syntax:**

```
$onVerbatim
$offVerbatim
```

These options are used in conjunction with the GAMS command line parameter DumpOpt to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.
Note

- The options $on/offVerbatim are only recognized for DumpOpt $\geq 10$ and apply only to lines in the file between the two options
- If $onVerbatim$ is active, DumpOpt = 11 behaves like DumpOpt = 21 (comments are kept)

Observe that the use of the options $goto$ and $on/offVerbatim$ are incompatible and may produce unexpected results.

Example:

```plaintext
$set f 123
$log %f%
$onVerbatim
$log %f%
$offVerbatim
$log %f%

The corresponding dmp file will contain the following lines:

$log 123
$onVerbatim
$log %f%
$offVerbatim
$log 123

See also command line parameter DumpOpt.
```

$[on][off]Warning  ($offWarning)

Syntax:

```
$onWarning
$offWarning
```

This option acts as a switch for data domain checking. In some cases it may be useful to accept domain errors in data statements that are imported from other systems and report warnings instead of errors. Data will be accepted and stored, even though it is outside the domain.

Attention

- This switch effects three types of domain errors usually referred to as error numbers 116, 170 and 171, see example below.
- This may have serious side affects and we recommend to exercise great care when using this feature.

Example:

```plaintext
Set i / one, two, three /
$onWarning
   j(i) / four, five /
   k / zero /;
Parameter x(i) "Messsed up Data" / one 1.0, five 2.0 /;
x('six') = 6;
x(j) = 10;
x('two') = x('seven');
j(k) = yes;
$offWarning
display i,j,x;
```
Note that the set \( j \), although specified as a subset of \( i \), contains elements not belonging to its domain. Similarly, the parameter \( x \) contains data elements outside the domain of \( i \). The skeleton listing file that results from running this code follows:

```plaintext
1 Set i / one, two, three /;
3     j(i) / four, five /
**** $170 $170
4 k / zero /;
5 Parameter x(i) "Messied up Data" / one 1.0, five 2.0 /;
**** $170
6 x('six') = 6; x(j) = 10; x('two') = x('seven');
**** $116,170
7 j(k) = yes;
**** $171
9 display i,j,x;
```

Error Messages

116 Label is unknown
170 Domain violation for element
171 Domain violation for set

**** 0 ERROR(S) 7 WARNING(S)

Execution

---- 9 SET i
one, two, three

---- 9 SET j
four, five, zero

---- 9 PARAMETER x Messied up Data
one 1.000, four 10.000, five 10.000, six 6.000

Observe that the domain violations are marked like normal compilation errors but are only treated as warnings and it is permitted to execute the code.

For an introduction to domain checking in GAMS, see section Domain Checking.

\$phantom id

Syntax:

\$phantom id

This option is used to designate id as a phantom set element. Syntactically, a phantom element is handled like any other set element. Semantically, however, it is handled like it does not exist. This is sometimes used to specify a data template that initializes the phantom records to default values.

Example:
$\text{phantom null}
\text{Set } i / \text{null } /
\qquad j / \text{a, b, null } /;
display i,j;

The output generated by the display statement is shown below:

\begin{verbatim}
---- 4 SET i
   ( EMPTY )
---- 4 SET j
   a, b
\end{verbatim}

Note that null does not appear in the listing file.

Attention

Statements that assign values to phantom labels are ignored.

Consider the following extension to the previous example:

$\text{Parameter } p(j) / \text{a 1, null 23 } /;
display p;

The output generated by the display statement is shown below:

\begin{verbatim}
---- 6 PARAMETER p
   a 1.000
\end{verbatim}

The system attribute system.empty is an implicitly defined phantom element. The following code works even without specifying $\text{phantom}$:

$\text{Set } i / \text{system.empty } /
\qquad j / \text{a, b, system.empty } /;
display i,j;

Another way to specify empty data statements makes use of on/offEmpty. The following example produces the same data as the data statement with the phantom label. In contrast to the example we $\text{phantom}$ we need to provide the dimensionality of the symbol $i$ explicitly via the $(*)$:

$\text{onEmpty}
\text{Set } i(*) / /
\qquad j / \text{a, b } /;
display i,j;

\$\text{prefixPath}

\text{Syntax:}

\$\text{prefixPath directoryPath}

This option augments the search path in PATH environment variable. The effect is that the text directoryPath is added to the beginning of the search path.

\text{Example:}
The log contains the following two relevant lines:

```
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1\Interacti
C:\somewhereelse\anotherpath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\Window
```

The option `setEnv` and `%sysEnv.VARNAME%` allow to modify system environment variables but the length of the environment variable value is limited in GAMS to 255 characters. The `PATH` environment variable is often much longer and therefore this special `$prefixPath` option exists.

This works on all platforms but the path separator depends on the operating system (`;` for Windows and `:` for Unix).

### $protect

**Syntax:**

```
$protect all | ident1 ident2 ...
```

This option creates a privacy setting: it freezes all values of identifiers with the result that modifications are no longer allowed but the parameters may still be used in model calculation (for example, equation definitions). Here `ident1` and `ident2` are specific GAMS identifiers previously defined in the program and the keyword `all` denotes all identifiers.

Note that this option is mainly used in the context of **secure work files**. The privacy restrictions may be removed with the options `$expose` or `$purge`.

### $purge

**Syntax:**

```
$purge all | ident1 ident2 ...
```

This option removes the identifiers and all associated data in a privacy setting. With explicit identifiers the listed identifiers are removed, and with `all` all identifiers are removed.

Note that this option is used in the context of **secure work files**. A special license file is needed for this feature to work, the removal only takes effect in the restart files.

### $remark

**Syntax:**

```
$remark text
```

This option performs a parameter substitution and writes a comment `text` to the compilation output of the listing file. Note that the line numbers of the comment are suppressed.

**Example:**

```
$set it TEST
$remark Write %it% to the listing file
```

The resulting listing file will contain the following line:
Write TEST to the listing file

$scratchFileName

Syntax:

$scratchFileName VARNAME [fileStem]

This option establishes or redefines the content of a scoped compile-time variable that is accessible in the code where the command appears and all code included therein. Here VARNAME is any user chosen variable name; fileStem is optional and used to built the name for a file in the scratch directory with a scratch extension (unless an extension is set explicitly). If it is omitted, the system will make up a random file name.

So,

$scratchFileName fn abc

is equivalent to

$set fn %gams.scrDir%abc.%gams.scrExt%

Example:

* Set fileStem explicitly
$scratchFileName fn1 abc
$log %fn1%

* Omit fileStem
$scratchFileName fn2
$log %fn2%

The log will show:

C:\Data\tmp\225a\abc.dat
C:\Data\tmp\__sFN__-0_313342928588555.dat

See also $set and section Compile-Time Variables.

$set

Syntax:

$set VARNAME text

This option establishes or redefines contents of a scoped compile-time variable that is accessible in the code where the command appears and all code included therein. Here VARNAME is any user chosen variable name; text is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that scoped compile-time variables may be destroyed (removed from the program) with the option $drop.

Note that in contrast to the option $eval the option $set does not evaluate the expression at compile time.

Note that GAMS allows scoped, local and global compile-time variables to be defined with the same name and therefore in some cases needs to prioritize. When referencing a compile-time variable via %VARNAME% a local variable hides scoped and global variables and a scoped variable hides the global variable as the following example demonstrates.

Example:
\$setLocal myvar this is a local variable
\$set myvar this is a scoped variable
\$setGlobal myvar this is a global variable
\$log %myvar%
\$droplocal myvar
\$log %myvar%
\$drop myvar
\$log %myvar%

The log will look as follows:

this is a local variable
this is a scoped variable
this is a global variable

If one wants to set a compile-time variable in an include file that is visible to the program after the $include one need to use $setglobal:

\$onEchoV > setvar.gms
\$setArgs varname varvalue
\$setglobal %varname% %varvalue%
\$offEcho
\$batInclude setvar MYVAR one
\$log %MYVAR%

The log will show

one

An inventory of all defined compile-time variables and their type (local, scoped, and global) is available with the option $show.

See also $setGlobal, $setLocal, and section Compile-Time Variables.

$setArgs

Syntax:

$setArgs id1 id2 id3 ...

With this option parameters that may be substituted are defined as GAMS compile-time variables. Note that $setArgs may only be used in external files that are included with the option $batInclude, $libInclude, and $sysInclude.

Example:

Scalar a /2/, b /4/, c /5/;
\$batInclude test3 a b c

The file test3.gms contains the following lines:

Scalar x;
x = %1 + %2 * %3 ;
display x;
$setArgs aa bb cc
x = %aa% - %bb% * %cc% ;
display x;
x = %1 + %2 * %3 ;
display x;
The option $setArgs$ allows the batInclude file to use the more descriptive compile-time variables $%aa%$ instead of $%1$, $%bb%$ instead of $%2$ and $%cc%$ instead of $%3$. Note that the use of $%1$, $%2$ etc. is still allowed. The program listing looks as follows:

```plaintext
1 Scalar a /2/, b /4/, c /5/;
BATINCLUDE C:\Users\default\Documents\gams\projdir\test3.gms
3 Scalar x;
4 x = a + b * c ;
5 display x;
7 x = a - b * c ;
8 display x;
9 x = a + b * c ;
10 display x;
```

and the output generated by the display statements follows:

```
---- 5 PARAMETER x = 22.000
---- 8 PARAMETER x = -18.000
---- 10 PARAMETER x = 22.000
```

See also $set$, $batInclude$.

$setComps$

Syntax:

```
$setComps perioddelimstring id1 id2 id3 ...
```

This option establishes or redefines compile-time variables so they contain the components of a period delimited string.

Here `perioddelimstring` is any period delimited string like the set specification of a multi-dimensional parameter, `id1` is the name of a scoped compile-time variable that will contain the name of the set element in the first position, `id2` is the name of a scoped compile-time variable that will contain the name of the set element in the second position and `id3` is the name of a scoped compile-time variable that will contain the name of the set element in the third position. The items may be recombined back into the original filename string by using $%id1%.%id2%.%id3%$.

Example:

```
$setComps period.delim.string id1 id2 id3
$log id1=%id1%
$log id2=%id2%
$log id3=%id3%
$set name %id1%.%id2%.%id3%
$log name=%name%
```

The resulting log file will contain the following lines:

```
id1=period
id2=delim
id3=string
name=period.delim.string"
```

See also $set$. 

$setDDList[.Cont]$
Syntax:

\>$setDDList\Cont id1 id2 \ldots$
\>$setDDList \quad id3 id4 id5 \ldots$

This option causes GAMS to look for misspelled or undefined double dash GAMS parameters.

**Example**: Consider the following example where three double dash GAMS parameters are defined on the command line:

```
> gams mymodel.gms --one=11 --two=22 --three=33 --four=44
```

The corresponding GAMS file follows:

```
$log \%one\%
$log \%two\%
$setDDList three
$log \%three\%
$log \%four\%
```

Note that the option `$setDDList three` checks if all double dash parameters have been used so far except for three. An error is triggered because four has not been used so far, the log file will contain the following:

```
*** 1 double dash variables not referenced
   --four=44
```

$setDDList\Cont$ can be used to define a list of parameters to be checked over multiple lines. So, the following two examples do the same thing:

```
$setDDList one two three four
$setDDList\Cont one
$setDDList\Cont two three
$setDDList four
```

Note

$setDDList\Cont$ does not do anything without a $setDDList$ statement.

See also section Double Dash Parameters.

**$setEnv**

Syntax:

```
$setEnv \ VARNAME \ value
```

This option defines an operating system environment variable. Here \VARNAME\ is a user chosen environment variable name and value may contain text or a number. Note that system environment variables are destroyed (removed from the program) with the option `$dropEnv` or when GAMS terminates.

**Example:**
The following output is echoed to the log file:

--- Starting compilation
this is very silly
this is very silly

See also $dropEnv and section Environment Variables in GAMS.

$setGlobal

Syntax:

$setGlobal VARNAME text

This option establishes or redefines contents of a global compile-time variable that is accessible in the code where the command appears and all code included therein and all parent files. Here VARNAME is any user chosen variable name; text is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that global compile-time variables may be destroyed (removed from the program) with the option $dropGlobal.

The difference between local, scoped, and global compile-time variable is explained with the option $set.

See also $set, $setLocal, $dropGlobal and section Compile-Time Variables.

$setLocal

Syntax:

$setLocal VARNAME text

This option establishes or redefines contents of a local compile-time variable that is accessible only in the code module (source file) where it is defined. Here VARNAME is any user chosen variable name; text is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that local compile-time variables may be destroyed (removed from the program) with the option $dropLocal.

The difference between local, scoped, and global compile-time variable is explained with the option $set.

See also $set, $setGlobal, $dropLocal and section Compile-Time Variables.

$setNames

Syntax:

$setNames file filepath filename fileextension
This option establishes or redefines three scoped compile-time variables so they contain the
drive subdirectory, filename and extension of a file named with full path. Here file is any
filename, filepath is the name of a scoped compile-time variable that will contain the name
of the subdirectory where the file is located, filename is the name of a scoped compile-time
variable that will contain the root name of the file and fileextension is the name of a scoped
compile-time variable that will contain the extension of the file.

Example:

```
$setNames "%gams.input%" filepath filename fileextension
$set name %filepath%%filename%%fileextension%
$log %name%
```

The log will show

```
C:\Users\default\Documents\gamside\projdir\ Untitled_1 .gms
```

Note that file is separated into its three components placing C:\Users\default\Documents\gamside\projdir\ into filepath, Untitled_1 into filename and .gms into fileextension. The three items may
be recombined back into the original filename by using %filepath%%filename%%fileextension%
as shown in the example.

If the file is missing a path, name, or extension the corresponding variable is defined but
remains empty as demonstrated in the following example:

```
$onEchoV > showfileparts.gms
$setNames "%1" filepath filename fileextension
$log path=%filepath%
$log name=%filename%
$log ext=%fileextension%
$offEcho
$batInclude showfileparts "C:\tmp\"
$batInclude showfileparts "Untitled_1"
$batInclude showfileparts "Untitled_1.gms"
$batInclude showfileparts "Untitled_1.gms.txt"
```

The log shows:

```
--- Untitled_1.gms(7) 2 Mb
path=C:\tmp\name=
    ext=
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(8) 2 Mb
    path=
    name=Untitled_1
    ext=.
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(9) 2 Mb
    path=
    name=Untitled_1
    ext=.gms
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(10) 2 Mb
    path=
    name=Untitled_1
    ext=.txt
```
Note that if a file contains multiple . the last one will be assigned to the fileextension as shown in the example with Untitled_1.gms.txt.

$shift

Syntax:

$shift

This option is similar to the command.com/cmd.exe shift operator (see \url{en.wikipedia.org/wiki/COMMAND.COM::Batch}). It shifts the order of all parameters passed once to the left. This effectively drops the lowest numbered parameter in the list.

Example:

Scalar a, b, c ; a = 1 ;
$batInclude inc.inc a b c
display a, b, c ;

The batch include file inc.inc follows:

%2 = %1 + 1 ;
$shift
%2 = %1 + 1 ;

The resulting listing file will contains the following echo print:

1 Scalar a, b, c ; a = 1 ;
BATINCLUDE C:\Users\default\Documents\gamsdir\projdir\inc.inc
3 b = a + 1 ;
5 c = b + 1 ;
6 display a, b, c ;

Note that in the first statement in the include file, %1 is the first argument in the $batInclude call and in this case it is interpreted as a. %2 is the second argument in the $batInclude call and is interpreted as b. This leads to the overall assignment being interpreted as b=a+1. The dollar control option $shift shifts the arguments to the left. As a result, %1 is interpreted as b, and %2 is interpreted as c. This leads to the second assignment being interpreted as c=b+1.

Therefore the outcome generated by the display statement in the input file is as follows:

---- 6 PARAMETER a = 1.000
       PARAMETER b = 2.000
       PARAMETER c = 3.000

See also $batInclude.

$show

Syntax:

$show

This option causes current values of the compile-time variables plus a list of the macros and active input and include files to be shown in the compilation output, where "active" means, that $show was either directly in that file or that file is a parent that includes the file with $show.

Example:
$set it 1
$setLocal yy
$setGlobal gg what
$include myinclude
$macro addx(x) x+x
$show

The file myinclude.gms follows:

$set inincs
$setLocal inincsl
$setGlobal inincsg
$macro multx(x) x*x
$show

The resulting listing file will contain the following environment reports in the compilation output:

---- Begin of Active File List
Level Type Line Filename
----------------------------------
1 INCLUDE 5 C:\Users\default\Documents\gamside\projdir\myinclude.gms
0 INPUT 4 C:\Users\default\Documents\gamside\projdir\Untitled_1.gms
---- End of Active File List

---- Begin of Compile-time Variable List
Level SetVal Type Text
----------------------------------
1 inincsl LOCAL
1 inincs SCOPED
0 yy LOCAL
0 it SCOPED 1
0 gg GLOBAL what
0 inincsg GLOBAL
---- End of Compile-time Variable List

---- Begin of Macro List
$macro multx(x) x*x
---- End of Macro List

and

---- Begin of Active File List
Level Type Line Filename
----------------------------------
0 INPUT 6 C:\Users\default\Documents\gamside\projdir\Untitled_1.gms
---- End of Active File List

---- Begin of Compile-time Variable List
Level SetVal Type Text
----------------------------------
0 yy LOCAL
0 it SCOPED 1
0 gg GLOBAL what
0 inincsg GLOBAL
---- End of Compile-time Variable List

---- Begin of Macro List
$macro multx(x) x*x
$macro addx(x) x+x
---- End of Macro List
4.41 Dollar Control Options

Note that only the macros and the item defined with the option $setGlobal in the included file carries over.

If one needs only parts of this report, the dollar control options $showFiles, $showMacros and $showVariables can be used.

See also section Compile-Time Variables.

$showFiles

Syntax:

$showFiles

This option prints the active files to the lst file as described at $show.

$showMacros

Syntax:

$showMacros

This option prints a list of macros to the lst file as described at $show.

$showVariables

Syntax:

$showVariables

This option prints a list current values of the compile-time variables to the lst file as described at $show.

$single

Syntax:

$single

The lines following this option will be echoed single spaced in the compilation output. Note that this is the default. The option is only useful as a switch to deactivate the option $double.

Example:

Set i / 1*2 / ;
Scalar a / 1 / ;
$double
Set j / 10*15 / ;
Scalar b / 2 / ;
$single
Set k / 5*10 / ;
Scalar c / 3 / ;

The echo print in the resulting listing file will look as follows:
$splitOption

Syntax:

$splitOption KEYVALPAIR optname optvalue

Establishes or redefines two scoped compile-time variables so they contain the name and value of an option key/value pair specified in various formats. KEYVALPAIR is a string formatted as -opt=val or -opt val (instead of - one can also use /). optname is the name of a scoped compile-time variable that will contain the name of the option and optvalue is the name of a scoped compile-time variable that will contain the value of the option. This is useful in particular in combination with batInclude files.

Example:

$onechoV > myinclude.gms
* Default values for named arguments
$setGlobal a1 1
$setGlobal a2 2
$setGlobal a3 nothing
$setLabel positionalArgs
$splitOption "%1" key val
$ if x%key%==x $goto FinishProcessNamedArguments
$ ifThenI.NamedArguments %key%==a1
$ setGlobal a1 %val%
$ elseIfI.NamedArguments %key%==a2
$ setGlobal a2 %val%
$ elseIfI.NamedArguments %key%==a3
$ setGlobal a3 %val%
$ else.NamedArguments
$ error Unknown named argument "%key%"
$ endIf.NamedArguments
$ shift
$goto ProcessNamedArguments
$setLabel FinishProcessNamedArguments
$setGlobal positionalArgs %1 %2 %3
$offEcho
$batInclude myinclude "-a3=some things" -a2=3.14 i j k
$log Using named arguments a1:%a1% a2:%a2% a3:%a3% positionalArgs:%positionalArgs%<

Now when calling this piece of code as a batInclude one can specify optionally some named arguments (in any order) right after the name of the batInclude file and before the positional arguments as demonstrated by the log output:
Using named arguments a1:>1< a2:>3.14< a3:>some things< positionalArgs:>i j k<

stars (****)

Syntax:

\$stars char[char][char][char]

This option is used to redefine the **** marker in the GAMS listing file. By default, important lines like those that denote errors and the solver and model status are prefixed with ****. A new marker consists of one to four characters.

Example:

\$stars *###*
garbage

The resulting listing file follows:

2 garbage
### $140
### $36,299 UNEXPECTED END OF FILE (1)

Error Messages
36 '=' or '..' or ':=' or '$=' operator expected
rest of statement ignored
140 Unknown symbol
299 Unexpected end of file

$sTitle

Syntax:

$sTitle text

This option sets the subtitle in the page header of the listing file to text. Note that the next output line will appear on a new page in the listing file.

Example:

$sTitle Data tables for input/output

See also $title.

$stop

Syntax:

$stop [text]

This option stops program compilation without creating an error. Note there is a difference to the option $exit. If there is only one input file, $stop and $exit will have the same effect. In an include file the option $exit acts like an end-of file on the include file. However, the option $stop in an include file will cause GAMS to stop reading all input but continue the execution phase of the so far compiled program. The text followed by $stop is ignored.

Example:
$ifthen not set EXPORTEXCEL
  $ stop No export to Excel
$else
  $ call gdxxrw ...
$endif

See also $abort, $error, $exit, and $terminate.

$sysInclude

The syntax of this dollar control option is equivalent to the syntax of $batinclude:

Syntax:

$sysinclude external_file arg1 arg2 ...

However, if an incomplete path is given, the file name is completed using the system include directory. By default, the system include directory is set to the GAMS system directory. Note that the default directory may be reset with the command line parameter sysIncDir.

Example:

The only relevant include file in the GAMS system directory is mpsgeset for MPSGE models, see for example [HARMGE]:

$sysInclude mpsgeset KAMIYA

Note that this call will first look for the include file [GAMS System Directory]/mpsgeset. If this file does not exist, it will looks for [GAMS System Directory]/mpsgeset.gms. The argument KAMIYA is passed on to the include file and are interpreted as explained for the dollar control option $batInclude.

Consider the following example:

$sysInclude C:\Users\default\Documents\mpsgeset KAMIYA

This call will first look specifically for the include file C:\Users\default\Documents\mpsgeset and next for C:\Users\default\Documents\mpsgeset.gms.

See also $batInclude.

$terminate

Syntax:

$terminate [text]

This option terminates compilation and also does not execution to program compiled so far without giving an error.

Example:

$if set JUSTTERMINATE $terminate

See also $abort, $error, $exit, and $stop.

$title

Syntax:
$title text

This option sets the title in the page header of the listing file to text. Note that the next output line will appear on a new page in the listing file.

Example:

$title Production Planning Model
$sTitle Set Definitions

See also $sTitle.

$unLoad

Syntax:

$unLoad [sym1[, sym2=gdxSym2[, ..., symN]]]

This option unloads specified items to a GDX file. Note that $unLoad must be used in conjunction with the option $gdxOut: $gdxOut must precede $unLoad. More than one option $unload may appear in between. Symbols can be renamed via the sym=GDXSym syntax. A $unLoad without arguments unloads the entire GAMS database into the GDX file.

Example: Consider the following slice of code:

Sets i 'canning plants' / seattle, san-diego /
   j 'markets' / new-york, chicago, topeka / ;

Parameters
   a(i) 'capacity of plant i in cases'
      / seattle 350
      san-diego 600 /
   b(j) 'demand at market j in cases'
      / new-york 325
      chicago 300
      topeka 275 / ;

Table d(i,j) 'distance in thousands of miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

$gdxOut tran
$unLoad i j=k
$unLoad b=dem a=sup
$unLoad d
$gdxout tranX
$unLoad

Note that the last lines will create a file named tran.gdx that contains i, j (now named k) and d and the parameters a and b which are now named dem and sup. The $unLoad in the very last line creates a GDX file tranX.gdx with all symbols (with their original names). The table of content (via $gdxIn and $load without parameters) of these two files looks as follows:
Both listings show domain information for the various symbols but only the file `tranX.gdx` created with `$unLoad` without arguments has consistent or full domain information while `tran.gdx` may have inconsistent or relaxed domain information. Domain matching when loading with the `$load sym<=[=]symGDX`, see `$load` for details, can be used in both cases but can become subtle with the relaxed domain information from `tran.gdx`.

Starting with GAMS 34 the behavior of `$gdxOut` and `$unload` changed compare to previous versions. In order to write consistent domain information and write aliases as such, the actual writing of the symbols to GDX is delayed until the GDX files is about to be closed either through an explicit `$gdxOut` (potentially with another filename) or through the end of the compilation phase. The advantage of the delayed writing is that the compiler can now better analyze the symbols and e.g. write alias symbols as aliases with the aliased set is also written to the GDX file (aliases were always written as sets before GAMS 34).

This new behavior can result in different GDX files. Here are two examples:

```
Set i /1,2,3/; Parameter p(i);
$gdxOut x.gdx
$unload
Parameter p(i) /1 1, 2 2, 3 3/;
```

In the GAMS versions before 34 the export GDX happened when the compiler processed the `$unload` command. At this point the parameter `p` did not have any data yet, so the GDX symbol `p` had no records. With the new behavior, the actual writing happens when the GDX file will be closed. Since there is no explicit close via a `$gdxOut` the GDX file is closed at the end of the compilation phase. By then the compiler has processed the statement `Parameter p(i) /1 1, 2 2, 3 3/;` and has filled `p` with data that will show up in the GDX file. In such a case an explicit `$gdxOut` after the `$unload` helps to get the old behavior back.

```
Set i /1,2,3/; Parameter p(i) / 1 1/;
$gdxOut x.gdx
$unload p=p1
$onmulti
Parameter p(i) / 2 2/;
$unload p=p2
```
In the GAMS versions before 34 the GDX file x.gdx contains two parameters p1 with record 1 1 and p2 with record 1 1, 2 2. With the new behavior the symbols are written again at the end of the compilation phase and by then the parameter p has the records 1 1, 2 2 and will write to GDX the parameters p1 and p2 with the two records each. The only way to accomplish the old behavior in this case is to write p1 and p2 to different GDX files:

Set i /1,2,3/; Parameter p(i) / 1 1/;
$gdxOut x1.gdx
$unload p=p1
$gdxOut x2.gdx
$onMulti
Parameter p(i) / 2 2/;
$unload p=p2

This delayed writing also influence other options like $onEpsToZero. The state of this becomes relevant when the file is actually written, not when a symbol is added to the list of symbols to be exported with $unload. This can be seen in the following example:

Set i / i1*i3 /;
Parameter p1(i) / i1 0, i2 EPS, i3 1 /
  p2(i) / i1 0, i2 EPS, i3 1 /;

$gdxOut out.gdx
$onEpsToZero
$unload p1
$offEpsToZero
$unload p2
$gdxOut

$call.checkErrorLevel gdxdump out.gdx

Here is the output from gdxdump at the end:

Parameter p1(*) /
  'i2' Eps,
  'i3' 1 /

Parameter p2(*) /
  'i2' Eps,
  'i3' 1 /

$use205

Syntax:

$use205

This option sets the GAMS syntax to the syntax of Release 2.05. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

$use205
Set if /1.2.3/;
Scalar x ;
The word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting option $use205 allows "if" to be used as an identifier since it was not a keyword in Release 2.05.

$use225

Syntax:

$use225

This option sets the GAMS syntax to the syntax of the first version of Release 2.25. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since the first version of Release 2.25. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

$use225
Set for /1.2.3/;
Scalar x ;

The word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting option $use225 allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.

$use999

Syntax:

$use999

This option sets the GAMS syntax to the syntax of the latest version of the compiler. Note that this setting is the default.

Example:

$use225
Set for /1.2.3/;
Scalar x ;
$use999
for (x=1 to 3, display x) ;

Note that the word "for" is used as a set identifier after setting the option $use225 and later the keyword for is used in a looping construct after having set the language syntax to that of the latest version using the option $use999.

$version

Syntax:

$version n

This issues a compilation error if n is greater than the current GAMS version. This can be useful to ensure that a model is run only with new versions of GAMS, because, e.g., a particular feature which did not exist in older versions is needed.

Example:
* With GAMS 24.8.1 the function numCores was added to the system.
* Make sure, that we use this GAMS version or newer.

\$version 24.8

Scalar nc "Number of cores";
nc = numCores;
Display nc;

\$warning

Syntax:

\$warning text

This dollar control option issues a compilation warning to the log and listing but continues
compilation and execution.

Example

\$ifthen not set INPUTFILE
$ set INPUTFILE default.txt
$ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite default.
$endif

The GAMS log file will issue a warning:

*** Error 332 in C:\Users\default\Documents\gamsdir\projdir\myinput.gms
   $Warning encountered - see listing for details

with the details in the listing file:

    3 $ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite default.
    **** $332

4.41 Dollar Control Options

GAMS offers several dollar control options that facilitate conditional compilation. In this section we will
first introduce the general syntax, present an overview of all relevant options and list the conditional
expressions that may be used to perform tests. Then we will give several examples to illustrate how these
options are used and to demonstrate their power. This section is meant as an introduction to conditional
compilation in GAMS and complements the detailed descriptions of the dollar control options listed in
Table 1 below.

4.41.4 Conditional Compilation

GAMS offers several dollar control options that facilitate conditional compilation. In this section we will
first introduce the general syntax, present an overview of all relevant options and list the conditional
expressions that may be used to perform tests. Then we will give several examples to illustrate how these
options are used and to demonstrate their power. This section is meant as an introduction to conditional
compilation in GAMS and complements the detailed descriptions of the dollar control options listed in
Table 1 below.

4.41.4.1 Conditional Compilation: General Syntax and Overviews

The dollar control option \$if and its variants provide a great amount of control over conditional processing
of the input file(s). The syntax in GAMS is similar to the IF statement of the DOS Batch language:

\$if [not] <conditional expression> new_input_line

The dollar control statement begins with \$if. Note that \$if may be replaced by one of its variants that
are listed in Table 1 below. The operator not is optional and makes it possible to negate the conditional
expression that follows. The conditional expression may take various forms, a complete list is given in
Table 2. The result of the conditional test is used to determine whether to process or not the remainder
of the line, new_input_line, which may be any valid GAMS input line.
Attention

The first non-blank character on the line following the conditional expression is considered to be the first column position of the GAMS input line. Therefore, if the first character encountered is a comment character the remainder of the line is treated as a comment line. Likewise, if the first character encountered is the dollar control character, the line is treated as a dollar control line.

Alternatively, the `new_input_line` may be placed in the next line. The corresponding syntax follows:

$$\texttt{$if \ [\text{not}] \ <\text{conditional expression}>}$

```
new_input_line
```

Note that in this version the space after the conditional expression is left blank. If the conditional is found to be false, either the remainder of the line (if any) will be skipped or the next line will not be processed.

The overviews in Table 1 and Table 2 conclude this subsection. Examples are given in the next subsection.

Table 1: `if` and Related Dollar Control Options

<table>
<thead>
<tr>
<th>Dollar Control Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if</code></td>
<td>This option is used to do case sensitive comparisons. Several examples are given in the next subsection.</td>
</tr>
<tr>
<td><code>ifE</code></td>
<td>This variant does the same as <code>if</code> but allows numerical constant expression evaluation. For an example, see the detailed description of this option.</td>
</tr>
<tr>
<td><code>ifI</code></td>
<td>This variant is the same as <code>if</code>, but it is case insensitive.</td>
</tr>
<tr>
<td><code>ifThen</code></td>
<td>This variant controls whether a block of statements will be processed or not. It is used to do case sensitive comparisons. Most often it is followed by one or more of the following dollar control options: <code>else</code>, <code>elseIf</code>, <code>elseIfI</code>, <code>elseIfE</code>. The option <code>ifThen</code> must be matched with the option <code>endIf</code> that marks the end of the block. An example is given below.</td>
</tr>
<tr>
<td><code>ifThenE</code></td>
<td>This is a variant of <code>ifThen</code> and is used for numerical comparisons. Like <code>ifThen</code>, it is often followed by the option <code>else</code> or one of its variants and must be matched with the option <code>endIf</code> that marks the end of the construct.</td>
</tr>
<tr>
<td><code>ifThenI</code></td>
<td>This is a variant of <code>ifThen</code> and is used to do case insensitive comparisons. Like <code>ifThen</code>, it is often followed by the option <code>else</code> or one of its variants and must be matched with the option <code>endIf</code> that marks the end of the construct.</td>
</tr>
<tr>
<td><code>endIf</code></td>
<td>This option must be matched with a preceding option <code>ifThen</code>, <code>ifThenE</code> or <code>ifThenI</code> and marks the end of the <code>if - then</code> construct. Note that the option <code>endIf</code> is not followed by a conditional expression, but it may be followed by a <code>new_input_line</code>. This GAMS input is restricted to other dollar control statements. An example is given below.</td>
</tr>
<tr>
<td><code>else</code></td>
<td>This option follows the option <code>ifThen</code>, <code>ifThenE</code> or <code>ifThenI</code>. It is followed by an instruction which is executed if the conditional expression of the matching <code>ifThen</code> statement is not true. Note that therefore this dollar control statement does not contain a conditional expression. An example is given below.</td>
</tr>
<tr>
<td><code>elseIf</code></td>
<td>This option follows the option <code>ifThen</code>, <code>ifThenE</code> or <code>ifThenI</code>. It is followed by another conditional expression and instruction. Note that this option is case sensitive. An example is given below.</td>
</tr>
<tr>
<td><code>elseIfE</code></td>
<td>This is a variant of <code>elseIf</code> that evaluates numerical values.</td>
</tr>
<tr>
<td><code>elseIfI</code></td>
<td>This is a variant of <code>elseIf</code> that is case insensitive.</td>
</tr>
</tbody>
</table>
Table 2: Conditional Expressions in Conditional Compilation

<table>
<thead>
<tr>
<th>Conditional Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acrType id</td>
<td>True if id is an acronym.</td>
</tr>
<tr>
<td>decla_OK</td>
<td>True if a declaration statement is permitted in the current line.</td>
</tr>
<tr>
<td></td>
<td>Note that declaration statements are not permitted within programming flow control structures like if statements or loop statements. An example is given below.</td>
</tr>
<tr>
<td>declared id</td>
<td>True if id was declared.</td>
</tr>
<tr>
<td>defined id</td>
<td>True if id was defined. An example is given below.</td>
</tr>
<tr>
<td>dExist directoryname</td>
<td>True if a directory with the name directoryname exists.</td>
</tr>
<tr>
<td>dimension n id</td>
<td>True if id has n dimensions. Note that n may take values from 0 to maximum number of possible indexes (see Dimensions).</td>
</tr>
<tr>
<td>equType id</td>
<td>True if id is an equation.</td>
</tr>
<tr>
<td>errorFree</td>
<td>True if compilation up to this point has been free of errors.</td>
</tr>
<tr>
<td>errorLevel n</td>
<td>True if the return code of a program called via $call is equal to or larger than n. For lists of GAMS return codes, see chapter GAMS Return Codes. An example is given below.</td>
</tr>
<tr>
<td>exist filename</td>
<td>True if a file with the name filename exists in the working directory or an input file directory. If no extension is specified, also filename.gms is checked. An example is given below.</td>
</tr>
<tr>
<td>filType id</td>
<td>True if id is the name of a put file.</td>
</tr>
<tr>
<td>funType id</td>
<td>True if id is a GAMS function.</td>
</tr>
<tr>
<td>gamsVersion n</td>
<td>True if current GAMS version is greater than or equal to n. GAMS versions are referenced with a single number. For example, if it should be tested whether the current GAMS version is 24.7 or newer, n will equal 247. Maintenance version numbers, e.g. 24.7.4 do not count.</td>
</tr>
<tr>
<td>gdxDimension n id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn and has n dimensions. Note that n may take values from 0 to maximum number of possible indexes (see Dimensions).</td>
</tr>
<tr>
<td>gdxEquType id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn and is an equation.</td>
</tr>
<tr>
<td>gdxParType id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn and is a parameter.</td>
</tr>
<tr>
<td>gdxSetType id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn and is a set.</td>
</tr>
<tr>
<td>gdxSymExist id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn.</td>
</tr>
<tr>
<td>gdxVarType id</td>
<td>True if id exists in a GDX file previously opened with $gdxIn and is a variable.</td>
</tr>
<tr>
<td>macType id</td>
<td>True if id is a macro.</td>
</tr>
<tr>
<td>modType id</td>
<td>True if id is a model.</td>
</tr>
<tr>
<td>onState key</td>
<td>True if the state of key is on (i.e. activated by $onKey). An example is given below.</td>
</tr>
<tr>
<td>parType id</td>
<td>True if id is a parameter.</td>
</tr>
<tr>
<td>preType id</td>
<td>True if id is a one of the predefined symbols in GAMS. Details are given below.</td>
</tr>
<tr>
<td>putOpen</td>
<td>True if both a file statement and at least one put statement have been compiled. Note that this does not guarantee that a file will be open at runtime.</td>
</tr>
<tr>
<td>readable id</td>
<td>True if id was correctly initialized, i.e. the symbol has a data statement or appeared on the left-hand side of an assignment statement, and may therefore be used on the right-hand side of an assignment statement. An example is given below.</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set varname</td>
<td>True if the scoped compile-time variable varname was set with the dollar</td>
</tr>
<tr>
<td></td>
<td>control option $set, $setGlobal or $setLocal.</td>
</tr>
<tr>
<td>setEnv varname</td>
<td>True if the environment variable varname was set in the systems environ-</td>
</tr>
<tr>
<td></td>
<td>ment, e.g. with the dollar control option $setEnv.</td>
</tr>
<tr>
<td>setGlobal varname</td>
<td>True if the control variable varname was set with the dollar control</td>
</tr>
<tr>
<td></td>
<td>option $setGlobal.</td>
</tr>
<tr>
<td>setLocal var_name</td>
<td>True if the control variable var_name was set with the dollar control</td>
</tr>
<tr>
<td></td>
<td>option $setLocal.</td>
</tr>
<tr>
<td>setType id</td>
<td>True if id is a set.</td>
</tr>
<tr>
<td>solver solver_name</td>
<td>True if a solver named solver_name exists in the GAMS system. An example</td>
</tr>
<tr>
<td></td>
<td>is given below.</td>
</tr>
<tr>
<td>uelExist id</td>
<td>True if unique element id exists in current GAMS database.</td>
</tr>
<tr>
<td>varType id</td>
<td>True if id is a variable.</td>
</tr>
<tr>
<td>warnings</td>
<td>True if the compilation until this point has been free of warnings.</td>
</tr>
<tr>
<td>xxxType id</td>
<td>True only if xxxType id is an unknown type. For more information, see</td>
</tr>
<tr>
<td></td>
<td>below.</td>
</tr>
<tr>
<td>string1 == string2</td>
<td>True only if string1 matches string2 exactly. Note that the strings may</td>
</tr>
<tr>
<td></td>
<td>be quoted or unquoted. Null (empty) strings may be indicated by an empty</td>
</tr>
<tr>
<td></td>
<td>quote: &quot;$&quot; or '&quot;'. The case of the strings provided either explicitly or,</td>
</tr>
<tr>
<td></td>
<td>more likely, through a parameter substitution, is preserved and</td>
</tr>
<tr>
<td></td>
<td>therefore will affect the string comparison. Quoted strings with leading</td>
</tr>
<tr>
<td></td>
<td>and trailing blanks are not trimmed and the blanks are considered part</td>
</tr>
<tr>
<td></td>
<td>of the string. Note that the string may have the form %VARNAME%, where</td>
</tr>
<tr>
<td></td>
<td>VARNAME refers to a compile-time variable including GAMS command line</td>
</tr>
<tr>
<td></td>
<td>parameters and system attribute. An example for a string comparison</td>
</tr>
<tr>
<td></td>
<td>with a command line parameter is given below. An example for a string</td>
</tr>
<tr>
<td></td>
<td>comparison with a system attribute is given below.</td>
</tr>
</tbody>
</table>

### 4.41.4.2 Conditional Compilation: Examples

#### 4.41.4.2.1 File Operation Test
The operator `exist` may be used to test whether a given file name exists. Consider the following example:

```
$if exist myfile.dat $include myfile.dat
```

Observe that the effect of this dollar control statement is that the file `myfile.dat` is included if it exists. Note that the character `$` at the beginning of the option `$include` is the first non-blank character after the conditional expression `exist myfile.dat` and therefore it is treated as the first column position. The statement above may also be written as follows:

```
$if exist myfile.dat
$include myfile.dat
```

#### 4.41.4.2.2 Conditional Compilation and Batch Include Files
In the next example we will illustrate how the option `$if` is used inside a batch include file where parameters are passed through the option `$batInclude` from the parent file:

```
$if not "%1a" == a $goto labelname
$if exist %1 file.ap=1;
```

Note that in the first line the `$if` condition uses the string comparison "%1a" == a to check if the parameter is empty. This test may also be done in the following way: %1 == "". If the parameter is not empty, the option `$goto` is processed.
4.41 Dollar Control Options

Note

The option $label cannot be part of the conditional input line. However, if the option $label appears on the next line, the condition decides once if the label is placed or not and subsequent instances of $goto will find the label without reevaluating the condition.

The second line illustrates the use of standard GAMS statements if the conditional expression is valid. If the file name passed as a parameter through the $batInclude call exists already, the GAMS will execute the $ap=1; statement which will append to the file.

The next example demonstrates how an unknown number of file specifications may be passed on to a batch include file that will include each of them if they exist. The batch include file could look as follows:

```
* Batch Include File - inclproc.gms
  * include and process an unknown number of input files
$label nextfile
  * Quote everything because file name might have blanks
$if exist "%1" $include "%1"
$shift
$if not "%1a" == a $goto nextfile
```

The call to this file in the parent file could take the following form:

```
$batInclude inclproc "file 1.inc" file2.inc file3.inc file4.inc
```

4.41.4.2.3 Testing Whether an Item Has Been Defined

The next example shows how to test if a named item was declared and/or defined.

```
Set i;
$if defined i $log First: set i is defined
$if declared i $log First: set i is declared
Set i /seattle/;
$if defined i $log Second: set i is defined
$if declared i $log Second: set i is declared
```

Note that after the first declaration of i only declared i evaluates to true when after the second declaration with a data statement both defined i and declared i are true.

4.41.4.2.4 Testing the state of a flag set by a dollar control option

The expression onState key tests whether the state of key is on. So the following example could be done with any key available as a $[on|off]Key dollar control option:

```
$if onState listing $log Listing is active at checkpoint 1
$if onState listing $set reactivateListing
$offListing
  * Do something which should not be seen in the listing
$if onState listing $log Listing is active at checkpoint 2
  * Reactivate listing if it was active at the start only
$if set reactivateListing $onListing
$if onState listing $log Listing is active at checkpoint 3
```
4.41.4.2.5 Testing Whether an Item May Be Used in an Assignment  The expression `readable` id tests whether data were assigned to an item and therefore the item may be used on the right-hand side of an assignment statement. Consider the following example:

Scalar f;
$if not readable f $log f cannot be used on the right
Scalar f /1/;
$if readable f $log f can be used on the right
$kill f
$if not readable f $log f cannot be used on the right after clear
f = 1;
$if readable f $log f can be used on the right after assignment

Note that in the first test the set f was declared, but there was no data statement, hence it is not readable. After a declaration with a data statement the test readable f evaluates to TRUE. With $kill we can revert f to a data less state and hence not readable f is TRUE after the "$kill". The assignment statement f = 1; make the scalar f readable again.

4.41.4.2.6 Testing Whether an Identifier May Be Declared  In programming flow control structures, like if statements or loop statements declaration statements are not permitted. The test `decla_ok` may be used to test whether the current environment allows declaration statements. Consider the following example:

$if decla_ok $log declarations are possible
if(1,
$ if not decla_ok $log declarations are not allowed
);

Note that the conditional expression in the both $if tests will evaluate to TRUE. However, the second test of decla_ok itself will be FALSE because it is processed while compiling an if statement, but with the not the entire expression evaluated to TRUE. For more information, see chapter Programming Flow Control Features.

4.41.4.2.7 Comments in the Context of Conditional Compilation  In-line and end-of-line comments are stripped out of the input file before processing the new_input_line. If either of these forms of comments appear, they will be treated as blanks. Consider the following example:

Parameter a ;
a=10 ;
$eolCom // inlineCom /* */
$if exist myfile.dat /* in line comments */ // end of line comments
a = 4 ;
display a;

Note that the comments on line 3 are ignored and the fourth line with the assignment statement will be processed if the conditional expression is true. Hence the outcome generated by the display statement will list a with a value of 4 if the file myfile.dat exists and a value of 10 if the file does not exist.
4.41.4.2.8 Error Level Test  Consider the following example:

\[
\texttt{
$\texttt{call gams mymodel.gms lo=2}$}
\texttt{
$\texttt{if errorlevel 1 $} \texttt{abort one or more errors encountered}$}
\]

Note that the errorlevel is retrieved from the previous system call via \texttt{$call$}. The conditional statement \texttt{errorlevel 1} is true if the returned errorlevel is equal to or larger than 1. In case of calling GAMS this means that something was not quite right with the execution of GAMS (either a compilation or execution error or other more exotic errors, see GAMS return codes. If this is the case, this GAMS program will be aborted immediately at compilation time.

Usually programs return 0 on success and non-zero on failure. The \texttt{if errorlevel 1} checks for strictly positive return codes. There are rare cases with failures and negative return codes (e.g. on Windows if some DLL dependencies of the program can't be resolved). In such a case \texttt{if errorlevel 1} will evaluate to false and not continue with the \texttt{abort} instruction. It might be better to access the program return code via the \texttt{errorLevel} function in the following way:

\[
\texttt{
$\texttt{call gams mymodel.gms lo=2}$}
\texttt{
$\texttt{ifE errorLevel<>0 $} \texttt{abort one or more errors encountered}$}
\]

4.41.4.2.9 Solver Test  The following example illustrates how to check if a solver exists.

\[
\texttt{
$\texttt{if solver ZOOM}$}
\]

Note that the conditional expression is false since the solver named \texttt{ZOOM} does not exists in the GAMS system (anymore).

4.41.4.2.10 Command Line Parameters in String Comparison Tests  Assume we include the following dollar control statements in a GAMS file called \texttt{myfile.gms}:

\[
\texttt{
$\texttt{if not '%gams.ps%'=='' $log Page size set to %gams.ps%}$}
\texttt{
$\texttt{if not '%gams.pw%'=='' $log Page width set to %gams.pw%}$}
\texttt{
$\texttt{if not '%gams.mip%'=='' $log MIP solver default is %gams.mip%}$}
\]

Then we run the program with the following call:

\[
> \texttt{gams myfile pageSize=60 pageWidth=85 mip=cbc}$
\]

Note that we specified values for the command line parameters \texttt{pageSize}, \texttt{pageWidth}, and \texttt{MIP}. We can either use the short or long name on the command line and in the compile-time variable. If we do not specify the option on the command line we will get the default value for option page size and page width. The MIP solver line will not show because \texttt{%gams.mip%} remains empty. The log with option setting on the command line will include the following lines:

\[
\text{Page size set to 60}
\text{Page width set to 85}
\text{MIP solver default is cbc}$
\]

Command line parameters are introduced in chapter The GAMS Call and Command Line Parameters.
4.41.4.2.11 System Attributes in String Comparison Tests

Compile-time system attributes may also be used in string comparison tests. The system attribute that is most useful in this context is .fileSys. It identifies the name of the operating system being used. Consider the following example:

```gams
$ifthen not %gams.logOption%==3
  $ ifi %system.fileSys%==UNIX  $set nullFile > /dev/null
  $ ifi %system.fileSys%==MSNT  $set nullFile > nul
$ if not set nullFile $abort %system.fileSys% not recognized
$else
  $ set nullFile
$endif
$call gamslib trnsport %nullFile%
```

These dollar control statements allow the definition of a NULL file destination that is dependent on the operating system that is being used. Note that the control variable nullFile contains an operating-system-dependent name. This is useful when making an external program that writes to STDOUT quiet in case the GAMS log does not go to STDOUT (logOption=3). This example could also have used the system attribute %system.nullFile% which contains the operating-system-dependent NULL file destination:

```gams
$set nullFile
$if not %gams.logOption%==3 $set nullFile > %system.nullFile%
$call gamslib trnsport %nullFile%
```

System attributes in general are introduced in chapter System Attributes.

4.41.4.2.12 Conditional Compilation with $ifThen and $else

Consider the following example which illustrates the use of $ifThen, $elseif, $else and $endif:

```gams
$set x a
$label test
$ifThen %x% == a $set x 'c' $log $ifThen with x=%x%
$elseif %x% == b $set x 'k' $log $elseif 1 with x=%x%
$elseif %x% == c $set x 'b' $log $elseif 2 with x=%x%
$else $set x 'e' $log $else with x=%x%
$endif $if not %x% == e $goTo test
```

Note that the resulting log file will contain the following lines:

```gams
$ifthen with x=a
$elseif 2 with x=c
$elseif 1 with x=b
$else with x=k
```

Observe that the options $else and $endif are not followed by conditional expressions and the instruction following the option $endif contains a dollar control statement. Moreover, note that the ‘$set x ‘’has the text to be set in quotes. GAMS needs to know where the text ends and the next dollar control option (in this case$log’) starts.
4.41 Dollar Control Options

4.41.4.2.13 Type of Identifiers  The type of a symbol can be retrieved via \$if ...Type. Consider the following example:

```
Set diag / 1*3 /;
Parameter p(diag) / 1 1, 2 4, 3 8 /;
$if setType diag $log diag is a set
$if not varType diag $log diag is not a variable
$if preType diag $log diag is a predefined type
$if parType p $log p is a parameter
$if setType sameAs $log sameAs is a set
$if preType sameAs $log sameAs is a predefined type
```

Note that for predefined symbols more than one type applies (e.g. sameAs is of set and predefined type). Please also note that diag is a set even though there is a predefined symbol named diag but that becomes invisible with a user defined symbols with the same name.

Normally there is no way to get a symbol into the GAMS symbols table without a proper type. However, if the dollar command line parameter multiPass is set to a value larger than zero, then the compiler will just check for some integrity and will try to deduce the symbol type from the context. If it is not able to do so, the symbol type will remain unknown. For example, compiling the following lines with multiPass=1

```
display x;
$if xxxType x $log x is of unknown type
```

result in the line x is of unknown type in the GAMS log.

4.41.5 Macros in GAMS

Macros are widely used in computer science to define and automate structured text replacements. The GAMS macro processors function similarly to the popular C/C++ macro preprocessor. Note that the GAMS macro facility has been inspired by the GAMS-F preprocessor for function definition developed by Michael Ferris, Tom Rutherford and Collin Starkweather, 1998 and 2005. The GAMS macro facility incorporates the major features of the GAMS-F preprocessor into the standard GAMS release as of version 22.9. GAMS macros act like a standard macro when defined. However, their recognition for expansion is GAMS syntax driven.

4.41.5.1 Syntax and Simple Examples

The definition of a macro in GAMS takes the following form:

```
$macro name  macro_body
$macro name(arg1,arg2,arg3,...) macro_body with tokens arg1, ...
```

The dollar symbol $ followed by macro indicate that this line is a macro definition. The name of the macro has to be unique, similar to other GAMS identifiers like sets and parameters. The macro name is immediately followed by a list of replacement arguments arg1, arg2, arg3,... that are enclosed in parentheses. The macro body is not further analyzed after removing leading and trailing spaces.

The recognition and following expansion of macros is directed by GAMS syntax. The tokens in the macro body to be replaced by the actual macro arguments follow the standard GAMS identifier conventions. Consider the following simple example of a macro with one argument:
Here the name of the macro is `reciprocal`, `y` is the argument and the macro body is `1/y`. This macro may be called in GAMS statements as follows:

```gams
$macro reciprocal(y) 1/y
scalar z, x1 /2/, x2 /3/;
z = reciprocal(x1) + reciprocal(x2);
```

As GAMS recognizes `reciprocal(x1)` and `reciprocal(x2)` as macros, the assignment statement will expand to:

```
z = 1/x1 + 1/x2;
```

Note that the macro use cannot be spread over multiple lines. Single line use with more complex arguments as in

```
z = reciprocal(x1+x2);
```

is recognized and will expanded to:

```
z = 1/x1+x2;
```

Note that `z` will be equal to 3.5 in this example as division takes precedence over addition. But breaking the argument of the macro over multiple lines as follows will trigger a macro expansion error:

```
z = reciprocal(x1
+x2);
```

The next example illustrates macros with multiple arguments:

```gams
$macro ratio(x,y) x/y
scalar z, x1 /2/, x2 /3/;
z = ratio(x1,x2);
```

The assignment above will expand to:

```
z= x1/x2;
```

Note that the macro definition may extend over several lines with the symbol `\` acting as a continuation string. Consider the following example:

```gams
$macro myxor(a,b) (a or b) \
            and (not a or not b)
scalar z;
z = myxor(1,0);
display z;
```

```
4.41 Dollar Control Options

The z assignment expands to

\[ z = (x_1 \text{ or } x_2) \text{ and } (\text{not } x_1 \text{ or } \text{not } x_2); \]

Note that although the macro has been defined over two lines, the expansion happens by combining the lines after stripping leading white spaces of the second line as demonstrated in the next example (because and has a higher precedence than or we can omit the parenthesis):

```gams
$macro myxor(a,b) not a and b \\
    or a and not b
scalar z;
z = myxor(1,0);
display z;
```

The z assignment expands to this:

\[ z = \text{not } 1 \text{ and } 0 \text{ or } 1 \text{ and } \text{not } 0; \]

The &, explained in more detail in the next section can be used to preserve (some of the) leading white spaces (but not the line breaks) if that is desired:

```gams
$macro myxor(a,b) not a and b \\
& or a and not b
```

4.41.5.2 Nested Macros

Macros may be nested. Consider the following example:

```gams
$macro product(a,b) a*b
$macro addup(i,x,z) sum(i,product(x(i),z))
set j / j1*j10 /;
Parameter a1(j) / #j 1 /, z, x1 /5/;
z = addup(j,a1,x1);
```

Observe that the macro product is nested in the macro addup. The assignment will expand to:

\[ z = \text{sum}(j,a1(j) \times x1); \]

Note that nested macros may result in an expansion of infinite length. An example follows.

```gams
$macro a b,a
display a;
```

This will expand into:

display b,b,b,b,b,b,b,b,b,b,b,b,...

In such a case GAMS will eventually refuse to do more substitutions and will issue a compilation error:

732 Too many edits on one single line - possible recursion in macro calls
compilation will be terminated
4.41.5.3 Ampersands in Macro Definitions

The expansion of arguments may be more carefully controlled by the use of ampersands & in the macro body. A single ampersand & is used as a concatenation or separation symbol to indicate tokens that are to be replaced. Consider the following example:

```
$macro f(i) sum(j, x(i,j))
$macro equ(q) equation equ_&q; \ 
equ_&q.. q =e= 0;
set i /i/, j /j/;
variable x(i,j);
equ(f(i))
```

This will expand into:

```
equation equ_f(i);equ_f(i).. sum(j, x(i,j)) =e= 0;
```

Note that without the ampersand notation, GAMS would have recognized only the third occurrence of q and hence the expansion would have been:

```
equation equ_q;equ_q.. sum(j, x(i,j)) =e= 0;
```

Two ampersands && immediately preceding a token will drop the most outer matching single or double quotes of the replacement argument. This makes it possible to include expressions with spaces, commas and unbalanced parentheses. The latter one is something users should really avoid doing. An example follows.

```
$macro d(q) display &&q;
$macro ss(q) &&q)
set i /i/, k /k/;
parameter a1(i) / i 1/, z;
d("here it is", i,k')
d("(zz")
z=ss('sum(i,a1(i)');
z=ss('prod(i,a1(i)');
```

Note that the expressions d contain quotes, spaces and commas and the expression ss has unbalanced parentheses within the quoted parts. In turn these expand to become:

```
display "here it is", i,k;
display "(zz";
z=sum(i,a1(i));
z=prod(i,a1(i));
```
4.41.5.4 Additional Macro Features

Deeply nested macros may require aliased sets in indexed operations like sum and prod. A minor syntax extension allows the implicit use of aliases. The suffix .local on a controlling set will use an implicit alias within the scope of the indexed operation. Consider the following example:

```plaintext
$macro ratio(a,b) a/b
$macro total(q) sum(i,q(i))
set i /i1*i15/;
parameter a(i) / #i 1 /, b(i) / #i 2 /, r(i), asum;
asum = total(a);
r(i) = ratio(total(a), b(i));
```

The assignment statement will expand to:

```plaintext
asum = sum(i,a(i));
r(i) = sum(i,a(i))/b(i);
```

The second line will not compile because the i in the sum is already controlled from the i on the left. The intention was the total macro is to add up the elements of a parameter indexed over i. As in the r(i) assignment the macro might be used in a statement where i is already controlled hence when doing the sum in the macro we want to use an alias of i. If we change the macro definition to

```plaintext
$macro total(q) sum(i.local,q(i))
```

The code works as expected because the i in the sum refers to the i.local and not the outside i.

Note that the the modifier .local is not limited to macros and may be used in any context. For further details and more examples, see the detailed description of the dollar command option $on/offLocal.

Another feature of macros is the implicit use of the suffix .L in report writing and other data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code illustrates this feature:

```plaintext
$macro sumIt(i,term) sum(i,term)
cost .. z =e= sumIt((i,j), (c(i,j)*x(i,j))) ;
supply(i) .. sumIt(j, x(i,j)) =l= a(i) ;
demand(j) .. sumIt(i, x(i,j)) =g= b(j) ;
Model transport /all/ ;
solve transport using lp minimizing z ;
Parameter tsupply(i) total demand for report
tdemand(j) total demand for report
$onDotL
tsupply(i)=sumIt(j, x(i,j));
tdemand(j)=sumIt(i, x(i,j));
```
The option \$onDotL enables the implicit suffix .L for variables. This feature was introduced for macros with variables to be used in equation definitions as well as assignment statements. The matching option \$offDotL will disable this feature. Similarly, \$offDotScale will access the .scale suffix of a variable or equation in an assignment statement.

Three more switches are relevant to macros. The option \$show will list any GAMS macros defined. The option \$on/\$offMacro will enable or disable the expansion of macros; the default is \$onMacro. Finally, the option \$on/offExpand will change the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch \$onExpand enables the recognition and expansion of macros in the macro argument list. The option \$offExpand will restore the default behavior.

Note that macro definitions are preserved in a save/restart file and are available again for a continued compilation.

Summarizing, macros shares the name space of GAMS symbols, like sets, parameters, variables, etc. Macros are recognized and expanded anywhere a proper GAMS identifier may be used. This may be suppressed with the option \$on/offMacro. The body of macros is only used during expansion. Hence, macro definitions are not order dependent. Variables in macro bodies will have an implicit suffix .L when they are used in assignment statements. This GAMS feature needs to be activated with the option \$onDotL.

### 4.41.6 Compressing and Decompressing Files

GAMS provides two dollar control options for compressing and decompressing GAMS input files:

<table>
<thead>
<tr>
<th>Dollar Control Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$compress &lt;source&gt; &lt;target&gt;</td>
<td>The file source is compressed into the packed file target.</td>
</tr>
<tr>
<td>$decompress &lt;source&gt; &lt;target&gt;</td>
<td>The compressed file source is decompressed into the unpacked file target.</td>
</tr>
</tbody>
</table>

**Attention**

Spaces are interpreted as separators between the source and target file names, hence quotes (single or double) have to be used if the file names contain spaces.

Note that GAMS will recognize whether a file is compressed and will processes it accordingly.

**Note**

Like any other GAMS input files, all compressed files are platform-independent.

#### 4.41.6.1 Compressing and Decompressing Files: A Simple Example

We use the well-known transportation model [TRNSPORT] to illustrate. First we copy the model from the GAMS Model Library and then we create a compressed version of the original:

```bash
> gamslib trnsport
> echo \$compress trnsport.gms t1.gms > t2.gms
> gams t2
```
Alternatively, the following code snippet may be used from within a GAMS file:

```
$call 'gamslib trnsport'
$compress trnsport.gms t1.gms
$include t1.gms
```

Note that the compressed input file `t1.gms` can be treated like any other GAMS input file. If it is executed, the listing file will be identical to the listing file of the original input file `trnsport.gms`, since a decompressed input is reported in the echo print. As usual, the parts of the model that are marked with the dollar control option `$on/offListing` will not appear in the echo print.

The compressed file `t1.gms` can be decompressed as follows:

```
> echo $decompress t1.gms t3.gms > t4.gms
> gams t4
```

Alternatively, from within a GAMS file:

```
$decompress t1.gms t3.gms
```

Observe that the decompressed file `t3.gms` is identical to the original file `trnsport.gms`. This can easily be tested with the following command:

```
> diff trnsport.gms t3.gms
```

### 4.41.6.2 Compressing and Decompressing Files: The Model CEFILES

The following more elaborate example is self-explanatory. It is adapted from model `[CEFILES]` and can easily be modified to test the use of compressed files.

```
* --- get model
$call gamslib -q trnsport

* --- compress and run model
$compress trnsport.gms t1.gms
$decompress t1.gms t1.org
$call diff trnsport.gms t1.org > %system.nullFile%
$if errorLevel 1 $abort files trnsport and t1 are different

* --- check to see if we get the same result
$call gams trnsport gdx=trnsport lo=%gams.lo%
$if errorLevel 1 $abort model trnsport failed
$call gams t1 gdx=t1 lo=%gams.lo%
$if errorLevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t1 are not equal

* --- also works with include files
$echo $include t1.gms > t2.gms
$call gams t2 gdx=t2 lo=%gams.lo%
$if errorLevel 1 $abort model t2 failed
$call gdxdiff trnsport t2 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t2 are not equal
$terminate
```
4.41.7 Encrypting Files

When models are distributed to users other than the original developers, issues of privacy, security, data integrity and ownership arise. To address these concerns, secure work files may be used and GAMS input files may be encrypted. Note, that the encryption follows the work file security model and requires special licensing.

Note

Like any other GAMS input files, all compressed and encrypted files are platform-independent.

Encryption is only available if a system is licensed for secure work files. There are two way to encrypt a file: Encryption for a particular user group that is identified by a target license file or for a general wide audience. In the prior case the key for encryption and decryption is extracted from the target license (specified by command line parameter pLicense) while for the latter the key for encryption and decryption is passed to GAMS via the command line parameters encryptKey and decryptKey. Note that once a file has been encrypted it cannot be decrypted any more. GAMS provides the following dollar control option to encrypt an input file:

$encrypt <source> <target>

Here the name of the input file to be encrypted is source and the name of the resulting encrypted file is target.

4.41.7.1 Encrypting Files: A Simple Example

We use again the transportation model [TRNSPORT] to illustrate. First we copy the model from the GAMS Model Library and then we create an encrypted version of the original via a target license:

> gamslib -q trnsport
> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 pLicense=target lo=%gams.logOption%

Note that the first two lines are similar to the directives that we have used to compress the model above. In the third line, the command line parameter pLicense specifies the target or privacy license to be used as a user key for encrypting. Thus the new encrypted file t1.gms is locked to the license key target and it can only be executed with the license file target:

> gams t1 license=target dumpOpt=11

Note that the command line parameter license is used to override the default GAMS license file gamslice.txt that can be located in various system wide and user locations. Note further that the command line parameter dumpOpt is usually used for debugging and maintenance. The value 11 causes a clean copy of the input to be written to the file t1.dmp, where all include files and macros are expanded. Observe that if some lines have been marked with the dollar control options $on/offListing in the original file, then these lines will be suppressed in the file t1.dmp.

An alternative approach to encrypt GAMS source file without a target license is done by the pair of command line parameters encryptKey and decryptKey:
In the third line, the command line parameter encryptKey specifies the encryption key. Thus the new encrypted file is locked and requires this key to successfully decrypt via command line parameter decryptKey:

> gams t1 decryptKey=ThisIsAPasswordSomeoneNeedsToRunTheModel

Any other key will result in a failure with the message Integrity check failed.

Note

Once a file has been encrypted, it cannot be decrypted any more. There is no inverse mechanism to recover the original file from the encrypted file. An attempt to decompress it using decompress will fail.

Observe that decrypting is done on the fly into memory when the GAMS system files are read. GAMS will recognize if a file is just plain text or compressed and/or encrypted and will validate and process the files accordingly.

4.41.7.2 Encrypting Files: The Model ENCRYPT

The following more elaborate example is self-explanatory; it is model [ENCRYPT] from the GAMS Model Library.

$title Input File Encryption Demo (ENCRYPT,SEQ=318)
$onText

Input files can be encrypted with an encryption key. Either the privacy license file mechanism or the command line parameter pair encryptKey/decryptKey can be for managing the password. Similar to compression, we offer an $encrypt utility to lock any file to a specific key. Once a file has been encrypted it can only be read by a gams program that has the matching license file or the matching decryption key. There is no inverse operation possible: you cannot recover the original GAMS file from the encrypted version.

To create an encrypted file, we need a license file which has the security option enabled.

Keywords: GAMS language features, input file encryption
$offText

$if not set MYPLICENSE $set MYPLICENSE "$gams.sysdir%license.txt"
$ifthen not exist "$MYPLICENSE%" $ log *** Target license file "$MYPLICENSE%" does not exist. $ log *** Specify via --MYPLICENSE=... $ log *** Encryption only via en/decryptKey $ drop MYPLICENSE $endif
* --- get model
$onDollar
$call gamslib -q trnsp

$ifthen.MYPLICENSE set MYPLICENSE
* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt trnsp.gms t1.gms > s1.gms
$call gams s1 plicense="%MYPLICENSE%" lo=%gams.lo%
$ifE errorLevel<>0 $abort encryption failed

$eolCom //
$if not errorFree $abort pending errors
$decompress t1.gms t1.org // this has to fail
$if errorFree $abort decompress did not fail
$clearError

*-- execute original and encrypted model
$call gams trnsp gdx=trnsp lo=%gams.lo%
$ifE errorLevel<>0 $abort model trnsp failed
$call gams t1 license="%MYPLICENSE%" gdx=t1 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t1 failed
$call gdxdiff trnsp t1 %system.reDirLog%
$ifE errorLevel<>0 $abort results for trnsp and t1 are not equal

* --- use the encrypted file as an include file
$onEcho > t2.gms
$offListing
* this is hidden
option limRow = 0, limCol = 0, solPrint = off;
$include t1.gms
$onListing
* this will show
$offEcho
$call gams t2 license="%MYPLICENSE%" lo=%gams.lo%
$ifE errorLevel<>0 $abort model t2 failed

* --- protect against viewing
* now we will show how to protect parts of an input
* file from viewing and extracting original source
* via the gams DUMPOPT parameter. We just need to
* encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 plicense="%MYPLICENSE%" lo=%gams.lo%
$ifE errorLevel<>0 $abort encryption failed
$call gams t3 license="%MYPLICENSE%" gdx=t3 dumpopt=19 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t3 failed
$call gdxdiff trnsp t3 %system.reDirLog%
$ifE errorLevel<>0 $abort results for trnsp and t3 are not equal

* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullfile%
$if not errorLevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullfile%
$if not errorLevel 1 $abort did not hide in dump file
4.41 Dollar Control Options

$endif.MYPLICENSE

* Same with encryptKey/decryptKey
* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt transport.gms t1.gms > s1.gms
$call gams s1 encryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE errorLevel<>0 $abort encryption failed

$eolCom //</p>
$if not errorFree $abort pending errors
$decompress t1.gms t1.org // this has to fail
$if errorFree $abort decompress did not fail
$clearError

*-- execute original and encrypted model
$call gams transport gdx=transport lo=%gams.lo%
$ifE errorLevel<>0 $abort model transport failed
$call gams t1 decryptKey=ThisIsMyPassword gdx=t1 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t1 failed
$call gdxdiff transport t1 %system.reDirLog%
$ifE errorLevel<>0 $abort results for transport and t1 are not equal

* --- use the encrypted file as an include file
$onEcho > t2.gms
$offListing
* this is hidden
option limRow = 0, limCol = 0, solPrint = off;
$include t1.gms
$onListing
* this will show
$offEcho
$call gams t2 decryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE errorLevel<>0 $abort model t2 failed

* --- protect against viewing
* now we will show how to protect parts of an input
* file from viewing and extracting original source
* via the gams DUMPOPT parameter. We just need to
* encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 encryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE errorLevel<>0 $abort encryption failed
$call gams t3 decryptKey=ThisIsMyPassword gdx=t3 dumpopt=19 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t3 failed
$call gdxdiff transport t3 %system.reDirLog%
$ifE errorLevel<>0 $abort results for transport and t3 are not equal

* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullfile%
$if not errorLevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullfile%
$if not errorLevel 1 $abort did not hide in dump file
4.42 The Put Writing Facility

4.42.1 Introduction

While the GDX facility is widely used to exchange bulk GAMS data with other programs the put writing facility allows to generate sophisticated reports in GAMS. The result are external ASCII files that are structured using information that is stored by the GAMS system. The put writing facility offers users numerous ways to control three format layers: the format of the external file that is written to, the format of the pages of the external files and the format of output items. Hence the structure of the put writing facility is more complex and requires more programming than the display statement, but there is much more flexibility and control over the output. The put writing facility generates external files automatically when the GAMS program is executed. The files are written sequentially, a single page at a time. The current page is stored in a buffer, which is automatically written to an external file when the page is full. Thus, the put writing facility has only control over the current page and does not have the ability to go back to alter former pages. However, while a particular page is current, information placed on it may be overwritten or removed at will.

This chapter is organized as follows. We will first introduce the file statement and the put statement, which are at the core of the put writing facility. Then we will present a simple example that will serve as illustration. We will also introduce some widely used features as we comment on the example. The remainder on the chapter will cover in detail the external report files, also called put files, the structure of put file pages and ways to control their format, types of output items and their formatting controls, and the put utility statement, a variant of the put statement that allows to special communication of formatted elements with the outside world. In addition, we will briefly discuss exception handling in the context of put statements and GAMS errors that are specific to put statements. We will conclude the chapter with an elaborate example. Note that put file attributes play a crucial role, since they are used for most formatting controls. A complete list of all file attributes is given in section Put File Attributes.

4.42.2 The Syntax

The basic structure of the put writing facility in its simplest form is as follows:

File file_name {,file_name};
put file_name;
put item {,item};

Note that the first line is a file statement. File statements define one or more external files that will be written to and specify internal names for them. These internal names will be used in the GAMS model to reference the external files when they are written to. The second line is a put statement that assigns the file with the name file_name as the current file. The third line is a put statement that writes one or more items to the current file. Items are any type of output like explanatory text, set labels, parameters, and variable, equation values and model attributes.

Next, we will present more details on the file statement and the put statement in the following two subsections and then we will turn to a first example that will illustrate how the the put writing facility works.
4.42 The Put Writing Facility

4.42.2.1 The File Statement: Defining Put Files

External files that are written to with the put writing facility are called put files. They are defined with a file statement. The syntax for a file statement is as follows:

```
File[s] file_name ["text"] [external_file_name]
    {,file_name ["text"] [external_file_name] } ;
```

The keyword `file` or `files` indicates that this is a file statement. It is followed by the internal name for the put file, `file_name`. The internal file name is a handle for the put file, it is used in the GAMS model to refer to the put file. The optional explanatory text may be used to describe the put file for future reference and to ease readability. The final part of the file statement is the actual name of the put file. Naming the external file is optional. In case it is omitted, by default, GAMS will create a name by appending the extension `.put` to the internal name. Consider the following example:

```
File results;
```

Note that in this statement no external file name is specified. Thus GAMS will create the external file `results.put` in the current working directory. Observe that by default, all put files are stored in the current working directory. There are several ways to specify alternative directories for put files. For details see section Choosing Where Put Files are Saved below.

Note

Multiple put files may be defined with one file statement.

Consider the following example:

```
File    class1
    class2   "this defines a specific external file" /report.txt/
    log      "this defines access to the GAMS log file" /''/;
```

Observe that the internal name of the first file is `class1`. As no external name is specified, GAMS will assign the absolute name `class1.put` to this file. The second file will be referenced in the model with the name `class2` and it corresponds to the external file `report.txt`. The third file is special: the internal name `''` is reserved for writing output to the GAMS log. Note that writing to the GAMS log can be useful to monitor how the solution process of the model is progressing.

For further details on put files, see section Put Files below.
4.42.2.2 The Put Statement

The put statement is at the core of the put writing facility. It has two different functions: it specifies which of the previously defined put files is the current file and it writes output to that file. The syntax for the first function is simple:

\[ \text{put file\_name;} \]

The keyword \text{put} indicates that this is a put statement, \text{file\_name} is the internal name of a put file that was previously defined with a file statement. This put statement has the effect that the specified file is now ready to be written to. Note that this put statement is necessary even if only one put file has been previously defined. For an example of how it is used when output is written to several different files, see the next section.

The basic syntax for a put statement that is used to write output to a file is as follows:

\[ \text{put item \{,item\};} \]

The statement begins with the keyword \text{put} followed by one or more items. Items may be a text (like a quoted text, an explanatory text, the name of a set element), a numerical value (like the value of a parameter, the value of an attribute, the solution status of the model) or a set value (\text{YES} or \text{NO} indicating whether a label is an element of a specific set). These items are discussed in detail in section Output Items below. In addition, GAMS facilitates writing a text block to a put file and including the content of an external file in a put file. For details see sections Text Items: Text Blocks and The Put Utility Statement respectively.

Note that it is also allowed to use only one put statement to assign one file as the current file and write to it. The syntax follows:

\[ \text{put file\_name item \{,item\};} \]

In addition, it is permitted to use just one put statement to write to multiple files sequentially. Thus the most general form of a put statement is as follows:

\[ \text{put file\_name item \{,item\} \{,file\_name item \{,item\}%;} \]

Note that only one file is current at a time. When a file is current, the output items following the name of this file will be written to the file. After this has been completed, the current file is reassigned to the next internal file name in the statement. The last internal file name used in a put statement continues to be the current file until a subsequent put statement uses an internal file name.

Observe that the keyword \text{put} has several variants: putclose, putltl, puthd, putpage and put_utility.
4.42 The Put Writing Facility

4.42.3 A First Example

We will use a small example to introduce the basic features of the put writing facility. The example is based on the well-known transportation model [TRNSPORT]. The following code segment could be placed at the end of the transportation model to create a report:

```gams
File factors /factors.dat/,
     results /results.dat/;
put factors;
put 'Transportation Model Factors' / /
   'Freight cost ', f,
   @1#6, 'Plant capacity'/;
loop(i, put @3, i.tl, @15, a(i)/);
put '/Market demand'/;
loop(j, put @3, j.tl, @15, b(j)/);
put results;
put 'Transportation Model Results' / /;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

In the first line, the file statement defines the internal file names `factors` and `results` and connects them to the external files `factors.dat` and `results.dat` respectively. These internal file names are used inside the model to reference files which are external to the model.

In the second line, the put statement assigns the file `factors` as the current file, that is the file which is currently available to be written to.

In the third line, the put statement starts the actual writing to the put file. The first item that is written is the quoted text string `Transportation Model Factors`. The item is followed by two slashes. A slash instructs the cursor to move to the first column of the next line. Two slashes have the effect that the cursor is moved to the first column of the second line, thus introducing a blank line.

Attention

Two slashes // are a popular end of line comment character sequence. So an intended blank line can result in a comment in the GAMS code. A safe way to use slashes in put statement is to separate the slashes by a space.

The next item is the string `Freight cost` followed by the value of the scalar `f`. Note that these output items are separated by commas. Blanks and commas serve as delimiters for separating different output items. These delimiters leave the cursor at the next column position in the file following the last item written. In most cases, a blank and a comma may be used interchangeably. However, the comma is the stronger form and will eliminate any ambiguities.

In the fifth line, the code above starts with the instruction @1#6. In the context of put statements, the symbols @ and # serve to reposition the cursor to a specific column and row respectively. Thus in our case, the cursor is repositioned to column 1 of row 6 of the put file. Then another text string is written and a new line is started. The semicolon terminates the put statement. More details on cursor positioning are given in section Controlling the Cursor On a Page.

In line 6, the next put statement is embedded within a loop statement. Note that the set `i` is the looping set. The put statement writes at column position 3 the set label name and at column position 15 the value of the parameter `a(i)` for each element of the set `i`. Observe that set element labels are referenced with the name of the set and the suffix `.tl`. For more information on identifier suffixes, see section Text Items below. Note that the put statement has to be placed within a looping structure, since only one element of the index set may be written with a put statement.
In line 7, the first symbol after the keyword put is a slash, that has the effect that a blank
line is inserted before the text string 'Market demand' is written.

In line 8, we have again a put statement within a loop structure: the values of the parameter b are written in a similar way to those of parameter a in line 6.

After execution, the put file factors.dat will look as follows:

Transportation Model Factors

Freight cost 90.00

Plant capacity
   Seattle 350.00
   San-Diego 600.00

Market demand
   New-York 325.00
   Chicago 300.00
   Topeka 275.00

Note that this output has been formatted using the default file format values. GAMS offers several ways to customize the format, see sections Global Item Formatting Controls and Local Item Formatting Controls below for further information.

In the last three lines of the code above, the file results.dat is made current and the level values associated with the variable x along with their corresponding set element index labels are written line by line. Note that the format of the output results of the variable x is customized by specifying a field width of 8 spaces with 4 of these spaces reserved for decimal places. This is an example of local formatting. The put file results.dat will contain the following lines:

Transportation Model Results

Seattle New-York 50.0000
Seattle Chicago 300.0000
Seattle Topeka 0.0000
San-Diego New-York 275.0000
San-Diego Chicago 0.0000
San-Diego Topeka 275.0000

This small example has demonstrated the main features of the put writing facility. However, its surface has just barely been scratched. In the remainder of this chapter we will describe in detail the many features of the put writing facility. Observe that in section Creating a Report for the Model MEXSS at the end of this chapter we will present a more elaborate report.

4.42.4 Put Files

As mentioned earlier, external files that are written to with the put writing facility are called put files. They are defined with a file statement and are made current with a put statement. Once they are current, they may be written to. By default, put files are saved in the current working directory. In this section we will cover more details on put files. We will discuss ways to specify other directories for put files, introduce the putclose statement, a variant of the put statement that closes the current file, and we will show how to append to an existing external file instead of replacing (overwriting) it. In addition, we will introduce the notion of put file attributes including a list of all attributes.
4.42 The Put Writing Facility

4.42.4.1 Choosing Where Put Files are Saved

Recall that by default, put files are saved in the current working directory. GAMS offers several ways to specify other directories for put files. The easiest way to specify another directory is by including the absolute or relative path in the file statement. Consider the following example:

```
File report / C:\Documents\GAMS\Output\report.dat /;
```

Note that the put file `report.dat` with the internal name `report` will be stored in the directory specified in the file statement instead of the current working directory.

An alternative directory for all put files in a model may be specified with the command line parameter `PutDir`. Assume we wish to write some output from running the well-known transportation model `[TRNSPORT]` to the file `results` defined above. Consider the following GAMS call:

```
> gams trnsport PutDir=c:\output
```

Note that the file `results.put` will be stored in the directory specified in the GAMS call. Observe that storing put files in the directory specified with `PutDir` is applies to all file that are not specified via an absolute file name. For more information on command line parameters, see chapter The GAMS Call and Command Line Parameters. scratchdir as putdir option (`.pdir`) lets GAMS write put files into the scratch directory.

The content of a put file will be sent to the GAMS log file if the put file is specified in the following way:

```
File name / '' /;
```

For example, the following code snippet may be added to the transportation model `[TRNSPORT]`:

```
File name / '' /;
put name ;
put 'instructions that will go to the GAMS log' /;
put 'more instructions that will go to the GAMS log' /
```

As a result, the respective GAMS log will contain the following lines:

```
... instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 01/25/17 15:46:09 elapsed 0:00:00.320
```
4.42.4.2 The Putclose Statement: Closing a Put File

Recall that the file statement and the put statement are at the core of the put writing facility. The putclose statement is a variant of the put statement, it is used to close a put file. The syntax is as follows:

```
putclose [file_name] {item} {file_name item {item}};
```

If the file_name is missing, putclose will close the current file. Often the putclose statement is used in this simple form. Consider the following example:

```
File report;
put report;
put "This is a report."
putclose;
```

Note that the last line has the effect that the file report.put is closed. The file may be used again later. As usual, it has to be assigned as the current file and then it may be written to. By default, an existing file is overwritten (replaced). Alternatively, it may be appended to. For details see section Writing to an Existing Put File below. The briefest version of the above code is

```
File report; putclose report "This is a report."
```

Closing a put file is useful for example when writing a solver option file from within the GAMS model. Consider the following example, where we will create and close an option file for the solver MINOS.

```
File opt 'Minos option file' / minos.opt /;
put opt 'Iteration limit 500' /
     'Feasibility tolerance 1.0E-7' /
putclose opt;
```

Note that the file minos.opt is closed with a simple putclose statement. This will make the file available to be used by the solver. Observe that the code snippet above has to be placed before the respective solve statement.

Now, the last three lines in the example above may be reduced to two lines in the following way:

```
putclose opt 'Iteration limit 500' /
     'Feasibility tolerance 1.0E-7' /
```

In this formulation, the putclose statement makes the file minos.opt current, writes to it and then closes the file after the last item has been written. Even though this is shorter, many users prefer the first formulation, since it is clearer.

Similarly to the put statement, the putclose statement may also be used to write to several files subsequently. The only difference to the put statement is, that the last current file will be closed after is has been written to.

Observe that GAMS automatically closes the put files of a model when it exits, even without a putclose statement.
4.42.4.3 Writing to an Existing Put File

When the put writing facility is used to write to a file that is not empty, by default, the existing content is overwritten (replaced). However, GAMS provides the option to append to the file instead via the file attribute `append option (.ap)`. Consider the following example:

```gams
File append /append.dat/
  put append;
  put "This is the first line." /;
  putclose;
  append.ap = 1;
  put append;
  put "This is the second line.";
```

Note that the `putclose` statement in line 4 closes the file `append.dat`. The assignment in the next line instructs the put writing facility to append to the file if the file is opened again and written to. Thus the following output is generated:

```
This is the first line.
This is the second line
```

For information on file attributes in general, see section Put File Attributes below.

4.42.4.4 Put File Attributes

Put files have attributes that are mainly used to customize the format of put files, put file pages and output items. Put file attributes are accessed in the following way:

```
file_name.attribute
```

Here `file_name` is the internal name of the put file and `.attribute` is the specific attribute that is to be accessed. We can also access the specific attribute of the current file by `file.attribute`. This useful in particular in `batincludes` that are used for multiple put files. Put file attributes may be used on the left-hand side and right-hand side of assignments. Consider the following example:

```gams
factors.nw = 10;
scalar x; x = factors.nw;
```

In the first line, the attribute `numeric field width (.nw)` of the put file with the internal name `factors` is assigned the value of 10. In the second line, the value of the attribute `.nw` is assigned to a scalar parameter `x`. In addition, put file attributes may be used as output items. For an example, see section Controlling the Cursor with File Attributes below. A complete list of put file attributes is given in Table 1.

**Table 1: Put File Attributes**
<table>
<thead>
<tr>
<th>Put File Attribute</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
<th>Optional Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append option</td>
<td>.ap</td>
<td>Allows to append to an existing file, instead of replacing (overwriting) it. For more information, see section Writing to an Existing Put File.</td>
<td>0</td>
<td>0: Overwrite 1: Append</td>
</tr>
<tr>
<td>Bottom margin</td>
<td>.bm</td>
<td>Number of blank lines to be placed in the bottom margin of the page. The bottom margin lines are in addition to the lines specified with the file attribute page size (.ps). Note that this attribute is functional only if the value of the suffix print control (.pc) is zero.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Alphabetic case</td>
<td>.case</td>
<td>Specifies the case in which alphabetic characters are displayed in the put file, regardless of the input. Note that the meaning of 1 and 2 differs from the one for lcase.</td>
<td>0</td>
<td>0: Mixed case 1: Upper case 2: Lower case</td>
</tr>
<tr>
<td>Current column</td>
<td>.cc</td>
<td>This attribute may be used to return or set the current cursor column in the main window. For more information, see Current Cursor Column.</td>
<td>0</td>
<td>Value may be between 1 and the page width.</td>
</tr>
<tr>
<td>Current row</td>
<td>.cr</td>
<td>This attribute may be used to return or set the current cursor row in the main window. For more information, see Current Cursor Row.</td>
<td>1</td>
<td>Value may be between 1 and the page size minus any header rows, title rows and margins.</td>
</tr>
<tr>
<td>Number of put errors</td>
<td>.errors</td>
<td>See section Put Errors for details and an example.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Header current column</td>
<td>.hdcc</td>
<td>This attribute may be used to return or set the current cursor column in the header block. For more information, see Current Cursor Column.</td>
<td>0</td>
<td>Value may be between 1 and the page width.</td>
</tr>
<tr>
<td>Put File Attribute</td>
<td>Symbol</td>
<td>Description</td>
<td>Default Value</td>
<td>Optional Values</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Header current row</td>
<td>.hdcr</td>
<td>This attribute may be used to return or set the current cursor row in the header block. For more information, see Current Cursor Row.</td>
<td>1</td>
<td>Value may be between 1 and the size of the header.</td>
</tr>
<tr>
<td>Header last line</td>
<td>.hdll</td>
<td>This attribute may be used to return the number of the last line of the page header or reset the last row written in the page header. For more information, see Last Line Control.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Alphabetic label case</td>
<td>.lcase</td>
<td>Specifies the case in which alphabetic characters of a label are displayed in the put file, regardless of the original casing.</td>
<td>0</td>
<td>0: Mixed case, 1: Lower case, 2: Upper case</td>
</tr>
<tr>
<td>Label justification</td>
<td>.lj</td>
<td>Alignment of set labels.</td>
<td>2</td>
<td>1: Right, 2: Left, 3: Center</td>
</tr>
<tr>
<td>Last line</td>
<td>.ll</td>
<td>This attribute may be used to return the number of the last line of the main window or reset the last row written in the main window. For more information, see Last Line Control.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Left margin</td>
<td>.lm</td>
<td>Number of empty columns to be placed on the left of the page.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Last page</td>
<td>.lp</td>
<td>Returns the number of pages that are already in the put file. Note that setting this attribute to zero does not delete the pages that have been written to the file.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label field width</td>
<td>.lw</td>
<td>Field width of set label output. For more information, see section Global Item Formatting Controls.</td>
<td>12</td>
<td>Values may be 0 or larger.</td>
</tr>
<tr>
<td>Put File Attribute</td>
<td>Symbol</td>
<td>Description</td>
<td>Default Value</td>
<td>Optional Values</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Number of decimals</td>
<td>.nd</td>
<td>Sets the number of decimals that are displayed for numeric items. A value of zero entails that fixed-format output will show the number rounded to the nearest integer. For an example, see section Global Item Formatting Controls.</td>
<td>2</td>
<td>Values may be between 0 and 20.</td>
</tr>
<tr>
<td>Numeric justification</td>
<td>.nj</td>
<td>Alignment of numeric output.</td>
<td>1</td>
<td>1: Right, 2: Left, 3: Center</td>
</tr>
<tr>
<td>Numeric round format</td>
<td>.nr</td>
<td>Selects the formatting used for numeric output. For example, one can choose to use scientific notation (i.e. E-format) for all numbers: this is especially useful for numbers so small that they display as only zeros in fixed format with the specified suffix number of decimals (.nd). For more information and an example, see section Global Item Formatting Controls.</td>
<td>1</td>
<td>0: Item is displayed in F or E format to fit given width and decimals. 1: Item is rounded to fit given width and decimals. 2: Item is displayed in E format to fit given width and decimals. 3: Item is rounded to fit given width. 4: Item is displayed in F or E format to fit given width.</td>
</tr>
<tr>
<td>Numeric field width</td>
<td>.nw</td>
<td>Field width of numeric output. For more information, see section Global Item Formatting Controls.</td>
<td>12</td>
<td>Values may be 0 or larger.</td>
</tr>
<tr>
<td>Numeric zero tolerance</td>
<td>.nz</td>
<td>Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding will be determined by the field width.</td>
<td>1.0e-10</td>
<td></td>
</tr>
<tr>
<td>PutDir becomes scratch directory</td>
<td>.pdir</td>
<td>Setting this to nonzero will result in resetting the PutDir to the scratch directory. Since the scratch directory is unknown while parameters are provided, this allows to use the scratch directory as PutDir.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Put File Attribute</td>
<td>Symbol</td>
<td>Description</td>
<td>Default Value</td>
<td>Optional Values</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Print control</td>
<td>.pc</td>
<td>Specifies the format of the put file. The options 4, 5, 6 and 8 create delimited files, which are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets. For an example, see section Creating a Report for the Model MEXSS.</td>
<td>2</td>
<td>0: Standard paging based on the current page size. Partial pages are padded with blank lines. Note that the file suffix bottom margin (.bm) is only functional when used with this print control option. 1: FORTRAN page format. This option places the numeral one in the first column of the first row of each page in the standard FORTRAN convention. 2: Continuous page. This option is similar to .pc option zero, with the exception that partial pages in the file are not padded with blank lines to fill out the page. 3: ASCII page control characters inserted. 4: Formatted output; non-numeric output is quoted and each item is delimited with a blank space. 5: Formatted output; non-numeric output is quoted and each item is delimited with commas. 6: Formatted output; non-numeric output is quoted and each item is delimited with tabs. 7: Fixed width, fills up line with trailing blanks. 8: Formatted output; each item is delimited with a blank space.</td>
</tr>
<tr>
<td>Put File Attribute</td>
<td>Symbol</td>
<td>Description</td>
<td>Default Value</td>
<td>Optional Values</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>-------------</td>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Page size</td>
<td>.ps</td>
<td>Number of rows (lines) that may be placed on a page of the put file. It may be reset by the user at any place in the program. Observe that the specification of this attribute is only effective if the attribute print control (.pc) has a value other than its default value. Note that a put error will result if it is set to value that is smaller than the number of rows which have already been written to the current page.</td>
<td>60</td>
<td>Any value between 1 and 130.</td>
</tr>
<tr>
<td>Page width</td>
<td>.pw</td>
<td>Number of columns (characters) that may be placed on a single row of the page. It may be reset by the user at any place in the program. Note that a put error will result if it is set to value that is smaller than the number of columns which have already been written to the current page. Observe that if a value is specified that is above the upper limit, then the value will be reset to the upper limit.</td>
<td>255</td>
<td>Any value between 1 and 32767.</td>
</tr>
<tr>
<td>Set value justification</td>
<td>.sj</td>
<td>Alignment of set values.</td>
<td>1</td>
<td>1: Right, 2: Left, 3: Center</td>
</tr>
<tr>
<td>Set value field width</td>
<td>.sw</td>
<td>Field width of set values. For more information, see section Global Item Formatting Controls.</td>
<td>12</td>
<td>Values may be between 0 and 20.</td>
</tr>
<tr>
<td>Put File Attribute</td>
<td>Symbol</td>
<td>Description</td>
<td>Default Value</td>
<td>Optional Values</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Text fill           | .tf    | Controls what content will be displayed, if there is no explanatory text for a set element, but a text item with suffix .te was specified. Note that options 4, 5 and 6 are useful if the output is intended to be included in a GAMS model at a later time. | 2             | 0: Blanks are displayed.  
1: Blanks will be displayed if the specified element does not exist, otherwise the name of the element will be displayed.  
2: The name of the element is displayed.  
3: The name of the element is displayed even if an explanatory text exists.  
4: Like option 3, but displays output in quotes with comma separators.  
5: Like option 4, but with periods as separators.  
6: Like option 4, but with blanks as separators. |
<p>| Text justification  | .tj    | Alignment of quoted text and explanatory text.                                                                                                                                                               | 2             | 1: Right, 2: Left, 3: Center                                                     |
| Title current column| .tlcc  | This attribute may be used to return or set the current cursor column in the title block. For more information, see Current Cursor Column.                                                                 | 0             | Value may be between 1 and the page width.                                       |
| Title current row   | .tlcr  | This attribute may be used to return or set the current cursor row in the title block. For more information, see Current Cursor Row.                                                                         | 1             | Value may be between 1 and the size of the header.                               |
| Title last line     | .tlll  | This attribute may be used to return the number of the last line of the page title or reset the last row written in the page title. For more information, see Last Line Control. | 0             |                                                                                  |</p>
<table>
<thead>
<tr>
<th>Put File Attribute</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
<th>Optional Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top margin</td>
<td>.tm</td>
<td>Number of blank lines to be placed at the top margin of the page. These lines are in addition to the number of lines specified with the file attribute page size (.ps).</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Text field width</td>
<td>.tw</td>
<td>Field width for quoted text and explanatory text. Note that a value of zero forces the field width to match the exact size of the item being displayed. For more information, see section Global Item Formatting Controls.</td>
<td>0</td>
<td>Value may be 0 or larger.</td>
</tr>
<tr>
<td>Window size</td>
<td>.ws</td>
<td>Returns the number of rows that may be written to the main window given the number of rows in the title and header blocks and the page size. Note that this attribute is computed by GAMS and cannot be changed by the user.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.42.5 Put File Pages

In this section we will discuss the structure of put file pages, in particular how to write titles and headers. In addition, we will give details on how users may control many aspects of the format of a put file page.

#### 4.42.5.1 Adding Titles and Headers

Titles and headers are widely used when there are sections of a page that remain relatively constant throughout a document. There are three independent writing areas on each page of a put file: the title block, the header block and the main window. The layout of a page is shown in the following diagram.

```
+-----------------+
| Title Block     |
+-----------------+
| Header Block    |
+-----------------+
| Main Window     |
```
Note that the put statement writes to the main window. There are two variants of the put statement that write to the title block and header block respectively: the puttl statement and the puthd statement.

Observe that every page must have an entry in the main window. If a page has no output in its main window, the page will not written, regardless of whether there are output items in the title or header blocks. To force a page that has an empty window out to file, we recommend to simply write something innocuous to the window, like the following:

```
put ''; // space
```

This will initiate the main window of the page and thus the page will be written.

Further, note that the size of any area of a page is based on the number of lines written to it and the total number of lines has to be smaller than the specified page size. A new page is started automatically whenever a page is full. For details on manual pagination, see section The Putpage Statement: Controlling Paging. Each area of a page is maintained independently. For example, we may write to the title block first, then to the header block and then again to the title block. However once we have written to the main window, all subsequent entries to the title and header are written to the next page, not the current page.

4.42.5.1.1 The Puttl Statement: Writing to the Title Block

The puttl (or putTitle which can be used as a synonym) statement is a variant of the put statement that writes to the title block of a page. The syntax is as follows:

```
puttl [file_name] item {item};
```

The keyword puttl indicates that this is a puttl statement. The internal name of the file, file_name, may be omitted if the desired file is already current. Like in the put statement, the content that is written is one or more items, that may be a text string, a numerical value or a set value. For more information on items, see section Output Items below. Consider the following example:

```
puttl factors 'GAMS Put Example';
```

If this line is placed before the put statement that writes to the main window in the example above, then GAMS Put Example will be written to the title block.

Observe that title blocks often contain the name of the model and the number of the put page. Once content is placed in the title block, it will be repeated on every page unless it is modified or deleted. For details on how to write output like the page number to a put file, see section System Suffixes as Output Items below.

Note

If content has already been written to the main window of a page, the items in the puttl statement will appear in the title block starting from the next page. Thus the title has to be written before any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the page size, the page will overflow resulting in a put error.
4.42.5.1.2  The puthd Statement: Writing to the Header Block  Like the puttl statement, the puthd (or putHeader which can be used as a synonym) statement is a variant of the put statement. Puthd statements write to the header block of a page. The syntax is as follows:

```
puthd [file_name] item {item};
```

The keyword puthd indicates that this is a puthd statement. The internal name of the file, file_name, may be omitted if the desired file is already current. Like in the put statement, the content that is written is one or more items. See section Output Items below for more information on items.

Observe that title blocks often contain a disclaimer or an instruction that is meant to be repeated on every page. Once content is placed in the header block, it will be repeated on every page unless it is modified or deleted.

Note

If content has already been written to the main window of a page, the items in the puthd statement will appear in the header block starting from the next page. Thus the header has to be written before any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the page size, the page will overflow resulting in a put error.

4.42.5.1.3  The putclear Statement: Deleting Title and Header Blocks  The putclear (or putfmcl which can be used as a synonym) statement may be used to delete the title and header blocks. The syntax is simple:

```
putclear;
```

If the keyword putclear is inserted after a title and/or header block has been written, but before the main window has been written to, then the title and/or header blocks of the current page will be deleted. If the main window has already been written to, the the title and/or header block of the next page and all subsequent pages will be deleted. Note that if the user wishes to delete either only the title or only the header block, the put file attributes .tlll and .hdll respectively may be set to zero. For more information, see Last Line Control.

4.42.5.2  The putpage Statement: Controlling Paging

A new page is started automatically if the bottom of a page is reached. Alternatively, a new page may be started early using a variant of the put statement: the putpage statement. The syntax is as follows:

```
putpage [file_name] {item};
```

The keyword putpage indicates that this is a putpage statement. In its simplest form, the putpage statement consists of just the keyword. If only the keyword is used, the current page will be terminated. The optional file_name makes the put file with the internal name file_name current. If output items follow, they will be written to the current page and a new page will be made available for the next put statement.

Consider the following example:

```
File example / example.txt /;
putpage example "This text is placed in the main window and the page ends here.";
put "Here starts a new page in the same putfile."
```

Observe that three file attributes are helpful for manual paging: last page (.lp), window size (.ws) and last line (.ll).
4.42 The Put Writing Facility

4.42.5.3 Controlling the Format of a Put File Page

While GAMS established gracious defaults, there are numerous ways to customize the format of a put file page. In this section we will present the put file attributes that facilitate the control of various aspects of page formatting and ways to control the position of the cursor.

4.42.5.3.1 Controlling Page Size, Page Width and Margins

By default, 60 rows (lines) may be written on a put file page. At any point in the program the file attribute page size (.ps) may be used to reset this value. Note that the upper limit is 130. Note further, that a put error will result if the attribute is set to value that is smaller than the number of rows which have already been written to the current page. Observe that the specification of .ps is only effective if the attribute print control (.pc) has a value other than its default value.

By default, 255 characters (columns) may be written to each line of a put file page. Like the page size, the page width may be reset at any point. GAMS provides the file attribute page width (.pw) to do this. Note that the upper limit is 32767. Note further, that a put error will result if the attribute is set to value that is smaller than the number of columns which have already been written to the current page.

By default, put file pages do not have top, bottom and left margins. However, it is easy to specify blank lines for the margins with the file attributes top margin (.tm) and bottom margin (.bm). Note that the lines reserved for the margins are in addition to the value specified for the page size (.ps). Observe that the file attribute .bm is only functional if the value of the file attribute print control (.pc) is set to zero. In addition, the number of empty columns for left margins may be specified with the file attribute left margin (.lm).

4.42.5.3.2 Controlling the Print Format and the Use of Capital and Lower Case Letters

By default, GAMS prints continuous pages. However, the file attribute print control (.pc) offers several alternative options. For a full list, see the respective entry in Table Put File Attributes above. Note in particular that the options 4, 5, 6 and 8 create delimited files, which are useful for importing output to other applications like spreadsheets. For an example, see section Creating a Report for the Model MEXSS.

By default, alphabetic characters are displayed in the case they were inputted. Note that the value of 1 for the file attribute alphabetic case (.case) will display all output in capital letters, regardless of the input, and the value of 2 will result in only lower case letters being displayed.

Assume that we wish a put file report.txt to have pages with 72 characters to a row and 58 lines, ASCII page control characters to be inserted at the end of every page, an additional top margin of 6 lines and all output displayed in upper case. The following assignments will implement these specifications:

```plaintext
File report /report.txt/;
report.pw = 72; report.ps = 58; report.pc = 3;
report.tm = 6; report.case = 1;
```

4.42.5.3.3 Controlling the Cursor On a Page

There are three ways to control the position of the cursor: with cursor control characters, with the system attribute system.tab and with file attributes. In this section we will give details on cursor control characters, while inserting tabs and file attributes that are relevant for cursor control will be discussed in the next two sections.

By default, the cursor is moved to the space immediately following the last character written. GAMS provides the control characters listed in Table 2 to specify another position for the cursor.
Table 2: Cursor Control Characters

Note that the numeric value n that follows the characters # and @ may be any expression or symbol with a numeric value. An example is given below. Note further, that the character @ is commonly used to align columns if the column headings have different widths. For an example, see section Creating a Report for the Model MEXSS. Observe that the character @ may be used to overwrite output items that were written earlier. Consider the following example:

```plaintext
File out /out.dat/;
put out;
put 'Good' @3 'morning everybody!';
```

This code snippet will generate the following output:

```
Good morning everybody!
```

Observe that cursor control characters are also used and discussed in the example at the beginning of this chapter.

4.42.5.3.4 Controlling the Cursor with a System Attribute: Inserting Tabs

The system attribute system.tab may be used to insert tabs. Consider the following example:

```plaintext
File test / test.dat /;
put test;
put "1" system.tab "2" system.tab "3" ;
```

The put file will contain the following line, separating 1, 2, and 3 by tabs.

```
1 2 3
```

4.42.5.3.5 Controlling the Cursor with File Attributes

In addition to cursor control characters and tabs, the position of the cursor may be controlled using the file attributes that will be discussed in this section. These file attributes refer to the current column, the current row and the last line in the title block, the header block and the main window.

Current Cursor Column

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current column. They are listed in Table 3.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#n</td>
<td>Move cursor position to row n of current page.</td>
</tr>
<tr>
<td>@n</td>
<td>Move cursor position to column n of current line.</td>
</tr>
<tr>
<td>/</td>
<td>Move cursor to first column of next line.</td>
</tr>
</tbody>
</table>
Table 3: File Attributes for the Column Position of the Cursor

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cc</td>
<td>Current cursor column in the main window.</td>
</tr>
<tr>
<td>.hdcc</td>
<td>Current cursor column in the header block</td>
</tr>
<tr>
<td>.tlcc</td>
<td>Current cursor column in the title block</td>
</tr>
</tbody>
</table>

Note that the values for these attributes are numeric and they are updated only at the end of a put statement. Consequently, their values will remain constant during a put statement, even if multiple items or lines are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current column position and return it respectively.

The following example illustrates the updating of the cursor control suffixes and the use of cursor control characters. The example is trivial but instructive:

Scalar lmarg 'left margin' /6/;
File out; put out;
put @(lmarg+2) 'out.cc = ', out.cc:0:0 ' ';
put @out.cc 'x'/ @out.cc 'y'/ @out.cc 'z ';
put 'out.cc = ' out.cc:0:0;

The resulting file out.put will contain the following lines:

```
out.cc = 1  x
      y  out.cc = 23
```

Note that the scalar lmarg is set to a specific value that will be used as an alignment tab. Symbols which hold common alignment values such as margins or tabs are often useful for large structured documents. The first put statement uses the cursor control character @ to relocate the cursor to column 8 where the text item out.cc and the respective value for the file attribute .cc are displayed. Note that the numerical value of the file attribute is formatted locally with the effect that only integers are displayed. We observe that at the start of the first put statement the file attribute out.cc equals 1.

The second put statement illustrates the updating of the cursor control suffixes by writing the letters x, y and z on three different lines. Each is preceded by moving the cursor to the value of the file attribute .cc. Note that at the end of the first put statement the value of out.cc is updated to 20. Hence, out.cc is 20 at the start of the second put statement. As a single put statement is used to write all three letters, the value of out.cc remains constant and thus the letters are written in the same column. At the end of the second put statement the value of out.cc value is updated to 23 (observe that there are two blank spaces after the letter z).

The third put statement writes the current value of the file attribute out.cc, which is the value at the start of the put statement.

Current Cursor Row

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current row. They are listed in Table 4.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cr</td>
<td>Current cursor row in the main window.</td>
</tr>
<tr>
<td>.hdcr</td>
<td>Current cursor row in the header block</td>
</tr>
<tr>
<td>.tlcr</td>
<td>Current cursor row in the title block</td>
</tr>
</tbody>
</table>
Table 4: File Attributes for the Row Position of the Cursor

Note that the values for these attributes are numeric and they updated only at the end of a put statement. Consequently, their values will remain constant during a put statement regardless of how many rows are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current row and return it respectively. The files attributes for the row position of the cursor behave similarly to those for the column position.

Last Line Control

Like the file attributes for the current column and row position, the file attributes for the last line have three variants, one for each writing area. They are given in Table 5.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ll</td>
<td>Last line in the main window</td>
</tr>
<tr>
<td>.hdl1</td>
<td>Last line in the header block</td>
</tr>
<tr>
<td>.tll1</td>
<td>Last line in the title block</td>
</tr>
</tbody>
</table>

Table 5: File Attributes for the Last Line

Note that unlike the row and column controls, the last line attributes are updated continuously. Last line attributes are especially useful for modifying the three writing areas of a page. They may be used on the left-hand side and on the right-hand side of an assignment statement to set the current last line and return it respectively. Rows will be deleted if the last line attribute is set to a value that is lower than the current row.

Attention

The file attributes .tll1 and .hdl1 may not hold values applicable to the current page, because when the title or header blocks are modified, they will correspond to the title or header blocks of the next page if the main window has been written to on the current page.

As mentioned above, in addition to determining the last line in a writing area, these file attributes may be used to delete lines within a writing area. In the following example, the header block will be completely deleted by resetting the attribute .hdl1 to zero.

File out;
puthd out 'This header statement will be eliminated';
out.hdl1 = 0;

Note that a header is written initially. However, by changing the attribute .hdl1 to zero, the cursor is reset to the top of the header block. As a result, the header will not be written unless something new is added to the header block.

4.42.6 Output Items

Output items are the items in the put statement that are written to the put file. They may be a text, a numerical value or a set value. In this section we will provide more information on these three types of items. Observe that system suffixes and command line parameters are either text strings or have a numerical value. They may also be used as output items.
4.42.6.1 Text Items

Text items may be a quoted text, an explanatory text of identifiers and labels or names of set elements. In this section we will give details on each of these text items. In addition, text items may be system attributes or command line parameters. For these text items, see sections System Suffixes as Output Items and Command Line Parameters as Output Items respectively. Details on default field widths and alignments for text items and global and local customizing controls are given in section Customizing the Format of Output Items.

4.42.6.1.1 Text Items: Quoted Text The simplest text output item is a quoted text. A quoted text is any combination of characters or numbers set apart by a pair of single (‘) or double (“) quotes with each the items needing to use a matching pair. Thus the following three lines are all exactly equivalent.

```
put 'Run on ' system.date ' using source file ' system.ifile;
put "Run on " system.date " using source file " system.ifile;
put 'Run on ' system.date " using source file " system.ifile;
```

Note that there is a limit on the length of all output items: output items may not exceed the page width. In case this limit is exceeded, a put error will be reported in the listing file. Put errors are introduced and discussed in section Put Errors.

4.42.6.1.2 Text Items: Identifier Attributes Text items like the explanatory text of an identifier, the name of a set element and the explanatory text of a set element are specified with identifier attributes. The identifier attributes are listed in Table 6.

<table>
<thead>
<tr>
<th>Identifier Attribute</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanatory text of an identifier</td>
<td>.ts</td>
<td>Displays the descriptive text associated with any identifier (like a set, a parameter, a variable, an equation, a model).</td>
</tr>
<tr>
<td>Name of a set element</td>
<td>.tl</td>
<td>Displays the names of the individual elements of a set. Observe that the put statement must be embedded in a loop statement, where the respective set is the looping set or the set is a singleton set.</td>
</tr>
<tr>
<td>Identifier Attribute</td>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Explanatory text of a set element</td>
<td>.te(index_list)</td>
<td>Displays the explanatory text associated with a set element. Note that this attribute requires the specification of a driving index. If the set is a simple set, the set name itself will have to be specified as index (see example below). If the set is defined over an index, then the index or a subset of the index may be specified. A special case arises in case users wish to gain access to the explanatory text that appears in the definition of another set (see example below). Observe that if there is no explanatory text for a set element in the set definition, then either the name of the set element (default) or a blank will be displayed, depending on the value of the file attribute text fill (.tf). Note that with .tf=3 even if an explanatory text exists, the name of the element will be displayed. Observe that the put statement must be embedded in a loop statement, where the respective set is the looping set or the set is a singleton set.</td>
</tr>
<tr>
<td>Indexed identifier with label combination</td>
<td>.tn</td>
<td>Displays the name of the identifier with all individual label combinations. Observe that the put statement must be embedded in a loop statement, where the respective index is the looping set or the set is a singleton set. See example below.</td>
</tr>
</tbody>
</table>

**Table 6: Identifier Attributes**

Consider the following example:

Set  
\[
i \text{ master set of sites } / i1 \text{ Seattle, } i2 \text{ Portland } \]  
\[
i3 \text{ San Francisco, } i4 \text{ Los Angeles, } i5 / \]  
\[
j \text{ subset of sites } / i3 * i5 / ; \]

File out; put out i.ts /;  
loop(i, put i.tl, i.te(i) /);

The resulting put file out.put will contain the following output:

```
master set of sites
i1 Seattle
i2 Portland
i3 San Francisco
i4 Los Angeles
i5
```
Note that the explanatory text of the set \( i \) is written first, followed by the label names of the set \( i \) and their respective explanatory texts. Note further that even though the set \( i \) does not have an index (it is a simple set), an index is required for the attribute \( .te \). In this case we specify as index the set itself. Observe that the set element \( i5 \) was defined without an explanatory text. By default, GAMS inserts the name of the element instead. This may be changed by altering the code in the following way:

```gams
put out i.ts /;
out.tf = 0;
loop(i, put i.tl, i.te(i) /);
```

Note that we assign the value of zero to the file attribute text fill (.tf) before the put statement where one of the items refers to the explanatory text of a set element. As a result, a blank will be displayed where the explanatory text of the set element \( i5 \) is supposed to appear:

```
master set of sites
i1    Seattle
i2    Portland
i3    San Francisco
i4    Los Angeles
i5
```

Assume we also wish to display the elements of the set \( j \) and their explanatory texts. We could just adapt the code above for the set \( j \) and add the following two lines:

```gams
put / j.ts /;
loop(j, put j.tl, j.te(j) /);
```

These lines will result in the following output to be appended to the put file:

```
subset of sites
i3
i4
i5
```

Note that the explanatory text associated with the set elements is missing, since there is no explanatory text for the elements in the definition of the set \( j \) and the value of zero for text fill is still valid. However, we may gain access to the explanatory text of the set elements in the definition of the set \( i \) in the following way:

```gams
put / j.ts /;
loop(j, put j.tl, i.te(j) /);
```

Note that the the specification \( i.te(j) \) singles out the explanatory texts of the elements of the set \( i \) that are also elements of the set \( j \). Hence we will obtain the desired output:

```
subset of sites
i3    San Francisco
i4    Los Angeles
i5
```
The following example illustrates the use of the identifier attribute .tn. The model [MEXSS] is a simplified representation of the Mexican steel sector, where five steel plants have to satisfy the demand for steel in three markets. Each steel plant \( i \) has several productive units \( m \) and the capacity of every unit in every plant is specified with the parameter \( k(m,i) \). Assume we wish to write to a put file a list of all nonzero capacities. We could add the following code to the model:

```plaintext
File out /out.dat/;
put out 'Capacity (in metric tons)' / /
loop((m,i)$k(m,i),
   put k.tn(m,i), @30 '=' k(m,i) /;
)
```

Note that we restrict the loop statement with a logical condition to exclude entries of zero. The put file out.dat will contain the following lines:

Observe that a more elaborate report for the model [MEXSS] is given in section Creating a Report for the Model MEXSS at the end of this chapter. The identifier attribute .tn is particularly useful when creating scalar EMP info files.

4.42.6.1.3 Text Items: Text Blocks The easiest way to write blocks of text to a put file is with the dollar control options $onPut and $offPut. Consider the following example:

```plaintext
File fx; put fx;
$onPut
We will write four
lines of text to the put file,

including a blank line.
$offPut
```

The put file fx.put will contain the following lines.

We will write four
lines of text to the put file,

including a blank line.

Note that these dollar control options have a variant that allows the substitution of compile-time variables. For details see the description of $on/offPut.
4.42.6.2 Numeric Items

Numeric items may be values of parameters, variable and equation attributes or model attributes and expressions of such elements. These numeric items will be discussed in this section. In addition, numeric items may be command line parameters. For these numeric items, see section Command Line Parameters as Output Items. Details on default field widths and alignments for numeric items and global and local customizing controls are given in section Customizing the Format of Output Items.

4.42.6.2.1 Numeric Items: Parameters and Functions  In our first example in this chapter, one of the output items was the parameter \( f \). For convenience, we repeat the respective code below:

\[
\text{put 'Freight cost ', f;}
\]

Note that the simple name of the parameter is sufficient. In the same example we also had two parameters that were defined over an index: \( a(i) \) and \( b(i) \). The respective lines of code follow:

\[
\text{loop(i, put @3, i.tl, @15, a(i)/;}
\]

* ...

\[
\text{loop(j, put @3, j.tl, @15, b(j)/;}
\]

Note that indexed parameters must be specified with their index and a put statement with an indexed parameter has to be embedded within a loop structure.

4.42.6.2.2 Numeric Items: Variable and Equation Attributes  Recall that data associated with variables and equations is stored in variable attributes and equation attributes respectively. A full list is given in sections Variable Attributes and Equation Attributes.

Suppose we wish to generate a report with the demand for each market in the transportation model [TRNSPORT], the satisfied demand after solution of the model and the marginal cost of meeting the demand. Recall that the demand data for each market is saved in the parameter \( b(j) \) and the relationship between shipment quantities and demand is encoded in equation \( \text{demand}(j) \). The following code will generate the desired report:

\[
\text{File report /report.dat/; put report;}
\]

\[
\text{loop(j,}
\]

\[
\text{put 'Report for ' j.tl/}
\]

\[
\text{'Demand' @35 b(j):10:0 /}
\]

\[
\text{'Demand satisfied' @35 demand.l(j):10:0 /}
\]

\[
\text{'Marginal Cost of meeting demand' @35 demand.m(j):10:2 /;}
\]

\]

Note that we use three numeric output items: the parameter \( b(j) \), the level value of the equation \( \text{demand}.l(j) \) and the marginal value of the equation \( \text{demand}.m(j) \). Note further, that all three items are indexed over the set \( j \) and thus the put statement has to be placed within a loop statement. Observe that we customize the formatting of the items. For details see section Local Item Formatting Controls. The put file report.dat will contain the following output:
Report for New-York
Demand 325
Demand satisfied 325
Marginal Cost of meeting demand 0.23

Report for Chicago
Demand 300
Demand satisfied 300
Marginal Cost of meeting demand 0.15

Report for Topeka
Demand 275
Demand satisfied 275
Marginal Cost of meeting demand 0.13

4.42.6.3 Numeric Items: Model Attributes GAMS models have many model attributes. An introduction and a complete list is given in section Model Attributes. While in principle all model attributes may be used as output items, the attributes .modelstat and .solvestat with their string valued counter parts .TModStat and .TSolStat are used most frequently. They refer to the model status and solver termination condition after solution respectively. For a complete list of their values, see sections Model Status and Solver Status.

Suppose we wish to generate a report that contains the time it took to execute the solve statement of the transportation model [TRNSPORT], the objective value and the corresponding string valued model and solver status. Recall that the name of the model is transport. The following code will generate the desired report:

File report /report.dat/; put report;
put 'elapsed time in seconds of solve statement : ' transport.etSolve:0 /
   'objective value: : ' transport.objVal:0 /
   'model status: : ' transport.tmodstat /
   'solver status: : ' transport.tsolstat ;

Observe that we customize the formatting of the first two items. For details see section Local Item Formatting Controls. The put file report.dat will contain the following output:

elapsed time in seconds of solve statement : 0.11
objective value: : 153.68
model status: : 1 Optimal
solver status: : 1 Normal Completion

4.42.6.3 Set Value Items

There are only two set values: YES and NO. The set value will be YES if a label is an element of a specific set and NO otherwise. Consider the following example which is adapted from the example in section Text Items: Identifier Attributes above:

Set i master set of sites / i1 Seattle, i2 Portland i3 San Francisco, i4 Los Angeles, i5 /
   j subset of sites / i3 * i5 / ;
File out2 / out2.dat /;
put out2 j.ts /
   out2.tf = 0;
loop(i, put i.tl, j(i), ' ', i.te(i) /);
Note that within the loop structure, the put statement writes each element of the set \( i \), determines whether it is a member of the set \( j \) and displays the respective set value, and adds the explanatory text for the label. Hence the resulting put file \( \text{out2.dat} \) will look as follows:

\[
\begin{align*}
\text{subset of sites} \\
i1 & \quad \text{NO Seattle} \\
i2 & \quad \text{NO Portland} \\
i3 & \quad \text{YES San Francisco} \\
i4 & \quad \text{YES Los Angeles} \\
i5 & \quad \text{YES}
\end{align*}
\]

Observe that the set values in the second column reflect set membership of the set \( j \). Observe further, that the missing explanatory text for label \( i5 \) is displayed as a blank, since the value of the file suffix \text{text fill} (.tf) equals zero.

### 4.42.6.4 System Suffixes as Output Items

System suffixes contain information about a GAMS run, they are introduced and discussed in chapter \text{System Attributes}. System suffixes may be used as output items in the context of put statements. They are accessed in the following two ways:

- \text{system.attribute}
- \%\text{system.attribute}\%

Here \text{system} is a keyword and \text{.suffix} is a specific system suffixes. Note that \text{system.suffix} references the \textit{execution-time} version (which can be of type string or numeric) of the system suffixes and \%\text{system.suffix}\% (which is interpreted by the compiler as part of the input string) references the \textit{compile-time} version. For further details on the difference between execution-time and compile-time system suffixes, see chapter \text{System Attributes}. A complete list of all system suffixes is given in section \text{List of all System Suffixes}, in Table 7 we present a list of the most common system suffixes.

<table>
<thead>
<tr>
<th>System Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.date</td>
<td>Program execution date</td>
</tr>
<tr>
<td>.ifile</td>
<td>Input file name</td>
</tr>
<tr>
<td>.ofile</td>
<td>Output file name</td>
</tr>
<tr>
<td>.rdate</td>
<td>Restart file date</td>
</tr>
<tr>
<td>.rfile</td>
<td>Restart file name</td>
</tr>
<tr>
<td>.sfile</td>
<td>Save file name</td>
</tr>
</tbody>
</table>
| .title           | Title of the model as specified by \\$\text{title}\

**Table 7: A Selection of System Suffixes**

To illustrate how system suffixes are used, assume we wish to include the program execution date, the name of the input (GAMS) file, and the page number of the current put statement in the input file to the report \text{results.dat} in section \text{A First Example} above. We will modify the code in the following way:

```plaintext
File factors /factors.dat/, results /results.dat/;
* ...
put results;
```
puthd 'Program Execution Date:', @26, system.date / 
'Source File:', @26, system.ifile / ;
'Page Number:', @26, system.page / ;
put 'Transportation Model Results' / ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);

The file results.dat will then contain the following lines:

Program Execution Date: 01/13/17
Source File: C:\Documents\GAMS\Models\transport.gms
Page Number: 1

Transportation Model Results
Seattle New-York 50.0000
Seattle Chicago 300.0000
Seattle Topeka 0.0000
San-Diego New-York 275.0000
San-Diego Chicago 0.0000
San-Diego Topeka 275.0000

Note that the date is given in the American format: month/day/year and can be reset via the command line parameter DFormat. Also note that including the page number in the header block of a put file will have the effect that the pages of the put file will be numbered. Of course, this is especially useful for longer put files.

It is also possible to read environment variables at execution time in a put context using sysEnv.name as in the following example:

$setEnv hello world
put_utility 'log' / sysEnv.hello;

If the environment variable queried is not defined, the result depends on the value of the command line parameter stringChk.

4.42.6.5 Command Line Parameters as Output Items

Command line parameters may also be used as output items. They are introduced and discussed in chapter The GAMS Call and Command Line Parameters. For an overview of all GAMS command line parameters, see section List of Command Line Parameters. Like system suffixes, command line parameters are referenced in the context of a put statement either compile-time strings or in an execution time version as follows:

put "%gams.parameter%";
put gams.parameter;

Here parameter is a GAMS command line parameter. To illustrate how command line parameters are used, assume we wish to include the page size of the input file, the name of the input file and the name of the restart file in the report results.dat in section A First Example above. We will modify the code in the following way:
File factors /factors.dat/, results /results.dat/ ;
* ... 
put results;
put 'Transportation Model Results'// ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 //);
put / "Page size = %gams.ps%
/ "GAMS input file = %gams.input%"
/ "GAMS restart file = %gams.restart%"

The last three lines of the put file results.dat follow:

Page size = 58
GAMS input file = C:/Documents/GAMS/Models/trnsport.gms
GAMS restart file =

Note that there was no restart file in the GAMS run, thus the value for %gams.restart% is the empty string.

In this example it actually makes no difference if one uses the compile- or execution time version to query the value of a command line parameter. This might be different, when doing the compile and execution phase separately, e.g. when using remote execution. Here is a toy example showing the difference:

$onEchoV > logScrDir.gms
put_utility 'log' / '%gams.scrDir%';
put_utility 'log' / gams.scrDir;
$offEcho

$call gams logScrDir.gms a=c s=1 keep=1
$call gams logScrDir.gms a=e r=1

In the log, one will see that %gams.scrDir% is evaluated in the first run at compile time already, while gams.scrDir is evaluated at execution time in the second run:

--- logScrDir.gms(1) 3 Mb
  C:\Data\tmp\225d\n--- logScrDir.gms(2) 3 Mb
  C:\Data\tmp\225e\n
4.42.6.6 Customizing the Format of Output Items

GAMS provides global and local controls to modify the default format of output items. Global controls are set with file attributes and apply to all output items in a put file that follow the assignment of a file attribute. Local controls are used to change the format of only one specific output item.

4.42.6.6.1 Global Item Formatting Controls The alignment (justification) of the field and the width of the field may be modified for all types of output items. The attributes that control field alignment are listed in Table 8. Note that possible values are 1 (right), 2 (left) and 3 (center).

<table>
<thead>
<tr>
<th>Type of Output Item</th>
<th>Symbol</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text items: set labels</td>
<td>.lj</td>
<td>2: Left</td>
</tr>
<tr>
<td>Text items: quoted and explanatory text</td>
<td>.tj</td>
<td>2: Left</td>
</tr>
<tr>
<td>Numeric values</td>
<td>.nj</td>
<td>1: Right</td>
</tr>
<tr>
<td>Set values</td>
<td>.sj</td>
<td>1: Right</td>
</tr>
</tbody>
</table>
Table 8: File Attributes for Field Alignment

The width of the field is specified with the number of spaces to be allocated. The attributes that control field width are listed in Table 9.

<table>
<thead>
<tr>
<th>Type of Output Item</th>
<th>Symbol</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text items: set labels</td>
<td>lw</td>
<td>12</td>
</tr>
<tr>
<td>Text items: quoted and explanatory text</td>
<td>tw</td>
<td>0</td>
</tr>
<tr>
<td>Numeric values</td>
<td>nw</td>
<td>12</td>
</tr>
<tr>
<td>Set values</td>
<td>sw</td>
<td>12 (maximum 20)</td>
</tr>
</tbody>
</table>

Table 9: File Attributes for Field Width

Note that the value of zero signifies a *variable* field width, matching the exact size of the item being displayed. If a text output item does not fit within the specified field, truncation will occur to the right. For numeric output items, the decimal portion of a number is rounded or scientific notation is used to fit the number within the given field. If a number is still too large, asterisks will replace the value in the output file.

For example, the following assignment will set the field width for numeric items in the file `out.put` globally to 4:

```
out.nw = 4;
```

In addition to field alignment and field width, there are further global controls that apply to numeric output items only. They are given in Table 10.

<table>
<thead>
<tr>
<th>File Attribute</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
<th>Optional Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of decimals</td>
<td>nd</td>
<td>Sets the number of decimals that are displayed for numeric items. A value of zero entails that fixed-format output will show the number rounded to the nearest integer.</td>
<td>2</td>
<td>Values may be between 0 and 20.</td>
</tr>
</tbody>
</table>
### File Attributes for Global Format Control Specific to Numeric Items

<table>
<thead>
<tr>
<th>File Attribute</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
<th>Optional Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric round format</td>
<td>.nr</td>
<td>Selects the formatting used for numeric output. For example, one can choose to use scientific notation (i.e. E-format) for all numbers: this is especially useful for numbers so small that they display as only zeros in fixed format with the value specified for number of decimals (.nd). The default rounding format will treat small values in this way, but in many situations, it is important to be aware that such small values exist.</td>
<td>1</td>
<td>0: Item is displayed in F or E format to fit given width and decimals. 1: Item is rounded to fit given width and decimals. 2: Item is displayed in E format to fit given width and decimals. 3: Item is rounded to fit given width. 4: Item is displayed in F or E format to fit given width.</td>
</tr>
<tr>
<td>Numeric zero tolerance</td>
<td>.nz</td>
<td>Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding is determined by the field width.</td>
<td>1.0e-10</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: File Attributes for Global Format Control Specific to Numeric Items

Note that numeric items are always formatted to fit within the specified width limit, but their display may use fewer characters. Similarly, for some values of .nr, the number of digits shown is limited by the choice of the .nd parameter, but fewer digits may be shown. At most 17 significant digits are included - this is enough to represent any double-precision value unambiguously, i.e. as a base-10 string that converts back into this same double. With this in mind, any number can be represented fully in E-format using at most 24 digits, e.g. -1.25E+102.

With .nr=0, the formatting routines honor the decimals limit. Values closer to 1 are displayed in fixed format. Values so small they would display only zeros in fixed format are displayed in E-format, as are values too large for the width limit. Formatting with .nr=1 is similar, but small values are shown in fixed format even if they show all zeros. With .nr=2 E-format is used for all values, with the number of trailing digits respecting the decimals limit .nd. The decimals limit is ignored for .nr=3, and fixed format is used unless the value is so large that E-format output is shorter. Finally, .nr=4 is intended to be used with larger output widths, e.g. 18 or 24, in order to show values of all magnitudes in as full a precision as possible: it ignores the decimal limit and uses whatever format allows more significant digits in the specified width.

The following example illustrates the result of different combinations of numeric file attributes. Note that we will use five combinations of the file attributes .nd, .nz, .nr and .nw to display three numerical values.

```bash
Set c suffix combinations / comb1 * comb9 /
```
v value indices / value1 * value5 / ;

<table>
<thead>
<tr>
<th>Table</th>
<th>suffix(c,*) numeric suffix combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nd nz nr nw</td>
</tr>
<tr>
<td>comb1</td>
<td>3 0 0 12</td>
</tr>
<tr>
<td>comb2</td>
<td>3 1e-5 0 12</td>
</tr>
<tr>
<td>comb3</td>
<td>3 1e-5 1 12</td>
</tr>
<tr>
<td>comb4</td>
<td>8 0 0 10</td>
</tr>
<tr>
<td>comb5</td>
<td>6 1e-5 1 12</td>
</tr>
<tr>
<td>comb6</td>
<td>0 1e-5 1 12</td>
</tr>
<tr>
<td>comb7</td>
<td>10 0 2 18</td>
</tr>
<tr>
<td>comb8</td>
<td>0 0 3 14</td>
</tr>
<tr>
<td>comb9</td>
<td>0 0 4 18</td>
</tr>
</tbody>
</table>

Parameter value(v) test values
/ value1 1.2345678901e14
value2 $\pi$
value3 -0.1234567
value4 0.0001234567
value5 1.234567e-13 / ;

File out; put out;
out.nj=2; out.lw=10; out.cc=11;
loop (v, put v.tl:21);
loop (c,
   out.nd=suffix(c,"nd");
   out.nz=suffix(c,"nz");
   out.nr=suffix(c,"nr");
   out.nw=suffix(c,"nw");
   put / c.tl;
   loop (v,
      put @(ord(v)*21-10), value(v)
   )
);

Observe that we have chosen to align the values to the left. This will enhance readability as the model loops through the suffix combinations that entail different field widths. The resulting output file out.put follows:

<table>
<thead>
<tr>
<th>value1</th>
<th>value2</th>
<th>value3</th>
<th>value4</th>
<th>value5</th>
</tr>
</thead>
<tbody>
<tr>
<td>comb1 1.2345678E+14</td>
<td>3.142</td>
<td>-0.123</td>
<td>1.2345670E-4</td>
<td>1.234567E-13</td>
</tr>
<tr>
<td>comb2 1.2345678E+14</td>
<td>3.142</td>
<td>-0.123</td>
<td>1.2345670E-4</td>
<td>0.000</td>
</tr>
<tr>
<td>comb3 1.2345678E+14</td>
<td>3.142</td>
<td>-0.123</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>comb4 1.2345678E+14</td>
<td>3.14159265</td>
<td>-0.1234567</td>
<td>0.00012346</td>
<td>1.234567E-13</td>
</tr>
<tr>
<td>comb5 1.2345678E+14</td>
<td>3.141593</td>
<td>-0.123457</td>
<td>0.000123</td>
<td>0.000000</td>
</tr>
<tr>
<td>comb6 1.2345678E+14</td>
<td>3</td>
<td>-0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>comb7 1.2345678901E+14</td>
<td>3.1415926536E+00</td>
<td>-1.2345670000E-01</td>
<td>1.2345670000E-04</td>
<td>1.234567E-13</td>
</tr>
<tr>
<td>comb8 1.23456789E+14</td>
<td>3.141592653590</td>
<td>-0.12345670000</td>
<td>0.00012345670</td>
<td>0.000000</td>
</tr>
<tr>
<td>comb9 123456789010000</td>
<td>3.14159265389739</td>
<td>-0.12345678</td>
<td>1.234567E-4</td>
<td>1.234567E-13</td>
</tr>
</tbody>
</table>

Note that in comb1 the display of values switches to the exponential notation when the value becomes smaller than the number of decimal places allowed. This is a result of .nr=0. Note further, that value4 is greater than the zero tolerance level (.nz), but smaller than the number of decimals allowed (.nd) in both, comb2 and comb3. However, .nr=0 results in the exponential notation in the display of comb2, while .nr=1 has the effect that this small value is rounded to zero. Observe that in comb6 fixed-format is used whenever allowed by the width limit, and in these cases the output is rounded to integer because .nd is
set to zero. With comb7 E-format is used in all cases: a leading space is added for positive values and a minus sign to negative values, and the magnitude is indicated with an explicit plus/minus and two digits (three if necessary). With comb8 we see fixed-format used whenever possible and as many digits (significant or not) as allowed by the width. With comb9 we see only significant digits, with the format chosen to allow the display of as many digits as possible.

### 4.42.6.6.2 Local Item Formatting Controls

The local item formatting controls allow to format specific output items. Note that local formatting overrides global format settings. The syntax is as follows:

```
item:{<>}width:decimals;
```

Here `item` is the output item, followed by a colon, an alignment (justification) symbol, the field width, a colon and the number of decimals to be displayed. Note that the limit on the number of the decimal places is only valid for numeric output. Note further, that if a component of the local formatting feature is omitted, then the respective global formatting settings will be used. The local alignment symbols are listed in Table 11.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Right</td>
</tr>
<tr>
<td>&lt;</td>
<td>Left</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Center</td>
</tr>
</tbody>
</table>

Table 11: Local Alignment Symbols

Observe that similar to global formatting, a field width of zero means that the field width will be variable, depending on the item to be displayed.

The following example serves as illustration. Observe that we use end-of-line comments to annotate the code.

```
File out; put out;
$eolCom //
Set   i / i1*i3 /;
Parameter d(i) / i1 1426, i2 1347, i3 900 /;
Scalar f / 17.6745 /;
loop(i, put d(i):0:0 /); // default justification and a field width
// of variable size with no decimals
put / 'Right Justified Comment':>50
// 'Center Justified Truncated Comment':<>20;
put / / f:<6:2; // left aligned scalar with 6 spaces for field width
// and two decimals
```

### 4.42.7 The Put Utility Statement

The put utility statement is a variant of the put statement that may be used to execute external programs. The syntax is as follows:
put_utility [file_name] 'command' / 'arguments' {/ 'command' / 'arguments'};

The keyword put_utility and its synonym put_utilities indicate that this is a put_utility statement. The keyword is followed by the internal name of an external file, file_name. Note that file_name may be omitted. It is not required for the put_utility statement but might be used to activate a file to be used with following put statements. Command denotes one of the commands listed in Table 12 below. Commands are followed by a slash and their respective arguments. Observe that a put_utility statement may contain multiple command / argument pairs. An example is given below.

The following simple example illustrates the put_utility statement:

File test / test.txt /;
put test "This is the original file."
put_utility 'ren' / 'test.dat';
putclos "This is the renamed file."

test.ap = 1;
put "Write to the renamed file."

Note that first the put file test.txt with the internal name test is defined, made current and written to. Then the put_utility statement uses the command ren to rename the current put file. The new name is test.dat and the internal name test will from now on reference the new file test.dat. However, the original file test.txt is not deleted. Hence, the code above will create two external files: test.txt and test.dat. The file test.txt will contain the following line:

This is the original file.

The file test.dat will have the following content:

This is the renamed file.
Write to the renamed file.

Note
If Put File Attributes are specified, they are also considered for the put_utility commands and arguments. This could lead to surprising results in some cases, like in this example:

File test / test.txt /;
test.pc = 5;
put test "This is the original file."

put_utility 'ren' / 'test.dat';
putclos "This is the renamed file."

This will trigger the following errors:

*** Error at line 5: Put_Utilities: Unknown request ""ren"
*** Error at line 5: Put_Utilities: Unknown request ""test.dat"

The problem is, that test.pc = 5; puts quotes around non-numeric output, and thus "ren" (including the quotes) is not recognized as command anymore. To overcome this issue test.pc should be set back to default for the ren command like this:
File test / test.txt /;
test.pc = 5;
put test "This is the original file."


test.pc = 2;
put_utility 'ren' / 'test.dat';
test.pc = 5;
putclose "This is the renamed file."

We will present a list of all commands and their arguments in Table 12 and give examples for most commands below.

Table 12: List of Commands and their Arguments

<table>
<thead>
<tr>
<th>Command</th>
<th>Description of Command</th>
<th>Description of Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignText</td>
<td>Allows to set the explanatory or label text of a singleton set element. See example below.</td>
<td>Name of singleton set symbol and text.</td>
</tr>
<tr>
<td>click</td>
<td>Adds a clickable file reference to the process window of the IDE. See example below.</td>
<td>File name of the file to which the reference will point.</td>
</tr>
<tr>
<td>dropEnv</td>
<td>Removes an environment variable at execution time.</td>
<td>Name of the environment variable.</td>
</tr>
<tr>
<td>ECArguments</td>
<td>Allows to set the arguments for execution time embedded code execution. See example in section Embedded Code at Execution Time.</td>
<td>Text for the embedded code argument.</td>
</tr>
<tr>
<td>exec</td>
<td>Passes a command to the operating system for execution. GAMS will wait until the command is executed. See example below. Note that the distinction between exec and shell is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.</td>
<td>Command to be executed with arguments.</td>
</tr>
<tr>
<td>exec.aSync</td>
<td>Passes a command to the operating system for execution. However, GAMS will not wait until the command is executed.</td>
<td>Command to be executed with arguments. Job control is handled identical to jobs spawned via execute.async.</td>
</tr>
<tr>
<td>exec.aSyncNC</td>
<td>Passes a command to the operating system for execution using a different console than the parent process (Windows only). GAMS will not wait until the command is executed, thus using multiple processors is possible. Job control is handled identical to jobs spawned via execute.async.</td>
<td>Command to be executed with arguments.</td>
</tr>
<tr>
<td>exec.checkErrorLevel</td>
<td>Passes a command to the operating system for execution. GAMS will wait until the command is executed, raises an execution error and aborts the execution, if the errorLevel returned is not 0.</td>
<td>Command to be executed with arguments.</td>
</tr>
<tr>
<td>Command</td>
<td>Description of Command</td>
<td>Description of Arguments</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>gdxIn</td>
<td>Accesses the GDX file specified in the argument. A subsequent directive <code>execute_load</code> or <code>execute_loadpoint</code> without a specified file name will unload data from the GDX file thus accessed. See example below.</td>
<td>Name of the GDX file that data will be loaded from.</td>
</tr>
<tr>
<td>gdxOut</td>
<td>Creates a new GDX file or accesses an existing GDX file specified in the argument. A subsequent directive <code>execute_unload</code> without a specified file name will write to the GDX file thus created or accessed. Note that if an existing GDX file is accessed, it will be overwritten. See example below.</td>
<td>Name of the GDX file to which data will be unloaded.</td>
</tr>
<tr>
<td>glb</td>
<td>Used by GAMS to facilitate building the model library - not intended for users.</td>
<td>–</td>
</tr>
<tr>
<td>htm</td>
<td>Used by GAMS to facilitate building the model library - not intended for users.</td>
<td>–</td>
</tr>
<tr>
<td>inc</td>
<td>Includes the contents of an external file in the currently active put file. See example below.</td>
<td>File name of external file.</td>
</tr>
<tr>
<td>incLog</td>
<td>Includes the contents of an external file in the currently log file. See example below.</td>
<td>File name of external file.</td>
</tr>
<tr>
<td>incMsg</td>
<td>Includes the contents of an external file in the currently listing file. See example below.</td>
<td>File name of external file.</td>
</tr>
<tr>
<td>incMsgLog</td>
<td>Includes the contents of an external file in both, the log file and the listing file. See example below.</td>
<td>File name of external file.</td>
</tr>
<tr>
<td>log</td>
<td>Sends a message to the log file. See example below.</td>
<td>Text of message.</td>
</tr>
<tr>
<td>msg</td>
<td>Sends a message to the listing file. See example below.</td>
<td>Text of message.</td>
</tr>
<tr>
<td>msgLog</td>
<td>Sends a message to both, the log file and the listing file. See example below.</td>
<td>Text of message.</td>
</tr>
<tr>
<td>ren</td>
<td>Creates a new external name for the current put file. The internal name will reference the new external file. Any subsequent put statements will write to the new file. See the simple example above and a more complex example below.</td>
<td>External file name.</td>
</tr>
<tr>
<td>save</td>
<td>Writes a save file of the current state of execution.</td>
<td>Name of save file.</td>
</tr>
<tr>
<td>setEnv</td>
<td>Sets an environment variable at execution time.</td>
<td>Name of the environment variable and its value.</td>
</tr>
</tbody>
</table>
4.42 The Put Writing Facility

<table>
<thead>
<tr>
<th>Command</th>
<th>Description of Command</th>
<th>Description of Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>shell</td>
<td>Passes a command to the command shell processor, where it is processed. The processed form of the command is then passed to the operating system for execution. See example below. Note that the distinction between shell and exec is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.</td>
<td>Command to be executed with arguments.</td>
</tr>
<tr>
<td>shell.checkErrorLevel</td>
<td>Passes a command to the command shell processor, where it is processed. The processed form of the command is then passed to the operating system for execution. GAMS checks the errorLevel returned implicitly, raises an execution error and aborts the execution, if is not 0.</td>
<td>Command to be executed with arguments.</td>
</tr>
<tr>
<td>solver</td>
<td>Selects a solver for a given or all model types (use *) by name. See example below.</td>
<td>Model type or * and solver name.</td>
</tr>
<tr>
<td>title</td>
<td>Changes the title on the DOS window.</td>
<td>New name for window.</td>
</tr>
<tr>
<td>winMsg</td>
<td>Sends a message to a window on a Windows machine. For examples, see models [ASYNNTRP] and [MRW01] in the GAMS Test Library.</td>
<td>Window name and message.</td>
</tr>
</tbody>
</table>

In the remainder of this section we will present examples.

**Exec: Creating Empty Files by Executing External Program touch**

Consider the following example:

```
Set i / 1*3 /;
loop(i, put_utility 'exec' / 'touch ' i.tl:0 '.txt');
```

Note that the command to be executed is `touch`. Thus this code snippet will create three empty files called `1.txt`, `2.txt` and `3.txt`.

**Exec and Ren: Creating Directories and Renaming Files**

Consider the following example:

```
File test / test.txt /; put test;
Set i / i01*i07 /;
loop(i,
    put_utilities 'exec' / 'mkdir ' 'test-':0 i.tl:0;
    put_utilities 'ren' / 'test-':0 i.tl:0 '%system.dirSep%test-':0 i.tl:0 '.txt':0 ;
    put 'this should be ' i.tl );
```

```
Observe that as the loop is executed, the first put_utility statement will create seven subdirectories called test-i01, ..., test-i07. The second put_utility statement will create a text file for each of the new subdirectories. The text file in subdirectory test-i01 is called test-i01.txt, the text file in subdirectory test-i02 is called test-i02.txt, etc. The put statement in the last line will write to each text file. For example, the following line will be generated for the text file test-i01.txt:

this should be i01

Note that at the end of the loop, the external file associated with the internal file name test is the put file test-i07.txt, since this file was the last current put file.

**Inc: Including the Content of a File**

In this example, we first create the external file recall.txt, write to it and close it. Then we create a new external file called report.dat. In a third step we include the content of the first file in the second file.

```plaintext
File recall /recall.txt/;
putclose recall "I am the external content."

File report /report.dat/;
put report "Here we include content from an external file.";
put_utility 'inc' / 'recall.txt' ;
```

Note that the file report.dat will contain the following lines:

Here we include content from an external file.
I am the external content.

**IncMsg, IncLog and IncMsglog: Including the Content of a File to the Log File and Listing file**

In this example, we first create the external file recall.txt, write to it and close it. Then we include the content of recall.txt into the log and listing file.

```plaintext
File recall /recall.txt/;
putclose recall "I am the external content." / "I go over more than one line."

display 'before the include';
put_utility 'incMsg' / 'recall.txt' ;
display 'after the include';
```

The listing file of this model shows the content of recall.txt:

```
---- 4 before the include
*** Start of include of file recall.txt
I am the external content.
I go over more than one line.
*** End of include of file recall.txt
---- 6 after the include
```
Msg, Log and Msglog: Writing to the Log File and Listing file

Consider the following example:

```gams
put_utility 'msg' / 'This message is for the lst file.'
   / 'log' / 'This message is for the log file.'
   / 'msgLog' / 'And this message is for the lst and the log file.' ;
```

Note that the following two lines will be generated in the log file:

This message is for the log file.
And this message is for the lst and the log file.

In addition, the listing file will contain the following two lines just before the report file summary.

This message is for the lst file.
And this message is for the lst and the log file.

Gdxout: Creating GDX Files and Unloading Data to them

Consider the following example:

```gams
Set j / 2005*2007 /;
Scalar random;
loop(j,
   put_utility 'gdxOut' / 'data' j.tl:0;
   random = uniform(0,1);
   execute_unload random;
);
```

This code will create the GDX files `data2005.gdx`, `data2006.gdx` and `data2007.gdx`. Note that each GDX file will contain a value between zero and 1 for the scalar `random`. For example, the file `data2005.gdx` (exported to an ASCII via `gdxdump`) will have the following content:

Scalar random / 0.171747132 /;

Gdxin: Loading Data from GDX Files

Note that the following example is an extension of the previous example that demonstrated the use of the command `gdxout`.

```gams
loop(j,
   put_utility 'gdxIn' / 'data' j.tl:0 ;
   execute_load random;
   display random;
);
```

This code loads the values of the scalar `random` from the GDX files `data2005.gdx`, `data2006.gdx` and `data2007.gdx` and displays them in the listing file of the GAMS input file:
Consider the following example:

Set j / j1*j5 /;
Scalar random;
loop(j,
    random = uniformint(0,100);
    put_utility 'shell' / 'echo ' random:0:0 ' > ' j.tl:0;
);

Observe that the shell script command `echo` outputs an integer between zero and 100 to the files j1, ..., j5.

Assume that there is a file called `sets.html` in our working directory. Consider the following code snippet:

```
put_utility 'click' / 'sets.html' ;
```

If we run this code snippet with the GAMS IDE, the following clickable link will appear in the IDE process window:

```
>>> File sets.html
```

### Solver: Select a solver by name at execution time

The `solver` keyword allows to select a solver at execution time by name. Normally, the option statement `option solver=xpress, lp=cplex;` is used to select the current solver, but in case where you want to programatically change the solver as in this example, this `put_utility` keyword can be useful. Consider the following code snippet which extends the [DICE] model:

```
... 
set slv 'MIP solvers to run' / cplex, gurobi, mosek, scip, xa, xpress /;
parameter rep 'report status, time, objective value, and more';
option bratio=1;
loop(slv,
    put_utility 'solver' / 'mip' / slv.tl:0;
    solve xdice using mip max wnx;
    rep(slv,'sstat') = xdice.solveStat;
    rep(slv,'mstat') = xdice.modelStat;
    rep(slv,'obj') = xdice.objVal;
    rep(slv,'time') = xdice.etSolve;
    rep(slv,'solver id') = xdice.sysIdent;
);
display rep;
```
If we run this code we get a report like this:

```
---- 82 PARAMETER rep report status, time, objective value, and more

<table>
<thead>
<tr>
<th>sstat</th>
<th>mstat</th>
<th>obj</th>
<th>time</th>
<th>solver</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplex</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>0.385</td>
<td>10</td>
</tr>
<tr>
<td>cbc</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>2.547</td>
<td>36</td>
</tr>
<tr>
<td>gurobi</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>1.077</td>
<td>36</td>
</tr>
<tr>
<td>mosek</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>2.372</td>
<td>56</td>
</tr>
<tr>
<td>scip</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>7.346</td>
<td>56</td>
</tr>
<tr>
<td>xa</td>
<td>3.000</td>
<td>8.000</td>
<td>21.000</td>
<td>20.061</td>
<td>80</td>
</tr>
<tr>
<td>xpress</td>
<td>1.000</td>
<td>1.000</td>
<td>21.000</td>
<td>1.460</td>
<td>81</td>
</tr>
</tbody>
</table>
```

If one want's to set a solver for all possible model types (similar to `option solver=xpress;`) specific model type has to be replaced by `*`: `put_utility 'solver' / '*' / 'xpress';`

**AssignText**: Assigns label text to an element of a singleton set

GAMS has no string data type. The explanatory or label text of an element can serve in several situations as a poor man's string type and used in `put` and `put_utility` statements. The example below extends the `TRNSPORT` model and disables flow on a particular connection. The solution point file created by `savePoint` gets renamed to e.g. `sol_seattle_new-york.gdx`. The filename is assembled using the `assigntext` keyword. This has been packaged for better readability in some STRING macros:

```
...$macro STRINGDEF(sym) singleton set sym / sym /
$macro STRING(sym) sym.te(sym)
$macro STRINGASSIGN(sym,text) put_utility 'assignText' / 'sym' / text
$macro STRINGAPPEND(sym,text) put_utility 'assignText' / 'sym' / sym.te(sym) text
alias (i,ii), (j,jj);
transport.savePoint = 1;
STRINGDEF(fname);
loop((ii,jj),
    x.up(ii,jj) = 0;
    solve transport min z using lp;
    x.up(ii,jj) = inf;
    STRINGASSIGN(fname,'sol');
    STRINGAPPEND(fname,'_' ii.tl:0);
    STRINGAPPEND(fname,'_' jj.tl:0 '.gdx');
    put_utility 'shell' / 'mv transport_p.gdx ' STRING(fname);
);```

### 4.42.8 Conditional Put Statements

In GAMS, shorthand notation for conditional statement are `dollar conditions`, as introduced in chapter Conditional Expressions, Assignments and Equations. Dollar conditions may also be used in the context of a `put` statement and its variants. The syntax in its most general form is as follows:

```
put $ logical_condition [file_name] item {item} {file_name item {item}};
```
Note that like all put statements, the conditional put statement begins with the keyword `put`. The keyword is followed by the dollar operator and a logical condition. If the logical condition is `TRUE` the put statement will be executed, otherwise the put statement will be ignored. For details on logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. Observe that the remainder of the conditional put statement is identical to the regular put statement introduced in section The Put Statement.

The following example demonstrates how dollar conditions are used in the context of put statements.

```gams
put$(a > 10) 'Some output items';
```

Note that the quoted text will only be written to the put file if the scalar or variable `a` is greater than 10.

### 4.42.9 Errors Associated with Put Statements

There are two types of errors that may occur when the put writing facility is used: syntax errors and put errors. In this section we will discuss these errors.

#### 4.42.9.1 Syntax Errors in Put Statements

Syntax errors are caused by the incorrect usage of the GAMS language, including unmatched parentheses, undefined identifiers, uncontrolled sets or the incorrect use of a keyword. These errors are detected during program compilation and are always fatal to program execution. For more information on compilation errors, see section Compilation Errors and the tutorials A GAMS Tutorial by Richard E. Rosenthal and Fixing Compilation Errors.

#### 4.42.9.2 Put Errors

Put errors are unique to the put writing facility. They are detected during program execution and are caused when the specifications of file attributes are violated. Typical errors include assigning inappropriate values to file attributes and attempts to write outside a page, like moving the cursor with the cursor control character `@` to a location beyond the page width. Consider the following example:

```gams
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";
```

Note that we specify the page width (`.pw`) to be just 8 characters. However, the quoted text has clearly more than 8 characters. In such a case the GAMS code will be compiled and the log file will report "Normal completion". At the appropriate position in the listing file the following error will be reported:

```
**** PUT ERROR FOR FILE out AT LINE 5: PUT LINE OVERFLOW - LOOK FOR **** ON PUTFILE, YOU CAN RESET .PW UP TO 32767
```

The put file `out.dat` will contain the following line:

`Let'****`

As put errors are not fatal and are not emphasized in the log file, they may be easily overlooked. Especially in large put files, put errors may go undetected. GAMS provides the file attribute `.errors` that facilitates the display of the number of put errors. Consider the following example:

```gams
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";
abort$(out.errors) "Put errors in out: ", out.errors;
```

Users may choose to output the number of put errors in a put file or display statement or even trigger an execution error as in the example above.
4.42 The Put Writing Facility

4.42.10 Creating a Report for the Model MEXSS

We started this chapter with a simple example and we will complete it with a more elaborate example. In this section we will show how the put writing facility may be used to create a report for the model [MEXSS]. The code for the report may be inserted at the end of the original model and is shown below in its entirety.

The model [MEXSS] analyzes the relative efficiency of five different plants for meeting the product requirements of ingot steel in three different markets. The model may be used to identify the major bottlenecks that constrain production in the system of plants. It will find the production levels in the steel mills and shipments from the mills to the markets that will meet the market requirements at least cost. We will create a report for this model that will present details on the available capacities of the productive units at the five plants, the unused capacities and the marginal values of the capacities. This report may be extended to include other data and results and the code may be reused to create new reports if the data in the model is changed. For models that are run periodically, say, weekly or monthly, reusable reports can be invaluable.

We will start the code for the report with defining the put file, setting the file attributes print control and page size and making the the put file current. Then we will specify some global formatting settings. We will continue with writing a title block and finally turn to the core of the report: a table with three subtables. Note that we will use in-line comments to annotate the code. The code follows:

```plaintext
$ compl //
File out /out.dat/;
out.pc=3; out.ps=54; // print control, page size
put out;

* Global Format Settings:
Scalars indent1 indent to first column of units display
       indent2 indent to first column of field labels
       indent3 indent to first column of first numeric field
       textwide wide text / 80 /
       textnarr narrow text / 30 /
out.nr = 0; // numeric round format
out.tw = textwide; // width of text field
out.lw = 11; // width of label field
out.nw = 11; // width of numeric field
out.nd = 2; // number of decimals displayed
indent1 = 3;
indent2 = 30;
indent3 = 27;

* Title Block
puttl 'MEXICO STEEL - SMALL STATIC MODEL':<> /
   system.date:<>/ /;

* Main Window
out.tj = 3; // alignment of text: center
put
   'This report is based on selected data and results from the model'/
   'MEXSS in the GAMS Library. This model analyzes the relative efficiency'/
   'of five different plants in meeting the product requirements for ingot'/
   'steel in three different markets. The model aims to find the pattern '/
   'of production levels in the mills and shipments from the mills to the '/
   'markets that will meet the market requirements at the least cost.' /
```


out.tj = 2; // alignment of text: left
out.tw = textnarr; // width of text field
putpage$(out.ll+card(m)+7 > out.ws); // manual paging

put 'Table 1. Plant Data and Results':0 '/--------'/; // Table 1
out.cc = indent2; // current column
loop(i, put i.tl::<>); // column headings

put / / 'CAPACITY (metric tons)'; // Capacity
loop(m,
    put / @indent1, m.te(m), @indent3; // row headings
    loop(i, put k(m,i)); // numeric values
);

* Header Block with column headings for next page (will be used only if necessary)
puthd 'Table 1 (continued). Plant Data and Results':0 /'----------'/;
out.hdcc = indent2;
loop(i, put i.tl::<>); // column headings
puthd '/';

* Main Window continued
if(out.ll+card(m)+sum((m,p)$b(m,p), 1)+3 > out.ws,
    putpage; // manual paging
    else put / /;
)

put 'UNUSED CAPACITY (metric tons)'; // Unused Capacity
loop(m,
    put / @indent1, m.te(m), @indent3; // row headings
    loop(i, put (k(m,i)-cc.l(m,i))); // numeric values
);

if(out.ll+card(m)+4 > out.ws,
    putpage; // manual paging
    else put / /;
)

put 'MARGINAL VALUE OF CAPACITY' @indent1 '(US$/ton)'; // Marginal Value of Capacity
loop(m,
    put / @indent1, m.te(m), @indent3; // row headings
    loop(i, put abs(cc.m(m,i))); // numeric values
);

Note that we set print control (.pc) to ASCII formfeed. Below we illustrate how to use one of the print control settings that generate a delimited file. Observe that we group all global format parameters and settings near the top of the code. This way of organizing the code will make it easy to modify the structure of the report in the future as needed. Observe further, that the output items in the title block are locally formatted to be center aligned.

In the remainder of the code the actual report is written in the main window. The block of text at the beginning serves as a brief introduction to the report. Note that the core of the report is a table consisting...
of three subtables, where the subtables share the column headings. Before writing the table, we insert a test - the conditional `putpage` statement - to determine whether there is a sufficient number of lines on the current page to accommodate the size of the first subtable. Thus the put writing facility would start a new page if there were not enough rows. We repeat similar tests before the code for the other two subtables. Observe that we insert a header block with the column headings. If a new page had to be started since there were not enough rows for the second or the third subtable, the header block would contain the column headings of the table. (Users might want to experiment by reducing the file attribute `page size (.ps)` to say, 30.) The report that will be generated follows:

**MEXICO STEEL - SMALL STATIC MODEL**

01/31/17

This report is based on selected data and results from the model MEXSS in the GAMS Library. This model analyzes the relative efficiency of five different plants in meeting the product requirements for ingot steel in three different markets. The model aims to find the pattern of production levels in the mills and shipments from the mills to the markets that will meet the market requirements at the least cost.


### Table 1. Plant Data and Results

<table>
<thead>
<tr>
<th>CAPACITY (metric tons)</th>
<th>ahsma</th>
<th>fundidora</th>
<th>sicartsa</th>
<th>hylsa</th>
<th>hylsap</th>
</tr>
</thead>
<tbody>
<tr>
<td>blast furnaces</td>
<td>3.25</td>
<td>1.40</td>
<td>1.10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>open hearth furnaces</td>
<td>1.50</td>
<td>0.85</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>basic oxygen converters</td>
<td>2.07</td>
<td>1.50</td>
<td>1.30</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>direct reduction units</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>electric arc furnaces</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.13</td>
<td>0.56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNUSED CAPACITY (metric tons)</th>
<th>ahsma</th>
<th>fundidora</th>
<th>sicartsa</th>
<th>hylsa</th>
<th>hylsap</th>
</tr>
</thead>
<tbody>
<tr>
<td>blast furnaces</td>
<td>0.13</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>open hearth furnaces</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>basic oxygen converters</td>
<td>0.00</td>
<td>0.72</td>
<td>0.14</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>direct reduction units</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
</tr>
<tr>
<td>electric arc furnaces</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.23</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MARGINAL VALUE OF CAPACITY (US$/ton)</th>
<th>ahsma</th>
<th>fundidora</th>
<th>sicartsa</th>
<th>hylsa</th>
<th>hylsap</th>
</tr>
</thead>
<tbody>
<tr>
<td>blast furnaces</td>
<td>0.00</td>
<td>69.62</td>
<td>71.69</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>open hearth furnaces</td>
<td>53.76</td>
<td>1.72</td>
<td>2.09</td>
<td>138.03</td>
<td>145.02</td>
</tr>
<tr>
<td>basic oxygen converters</td>
<td>64.57</td>
<td>0.00</td>
<td>0.00</td>
<td>138.03</td>
<td>145.02</td>
</tr>
<tr>
<td>direct reduction units</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>80.08</td>
<td>0.00</td>
</tr>
<tr>
<td>electric arc furnaces</td>
<td>136.46</td>
<td>138.03</td>
<td>140.00</td>
<td>0.00</td>
<td>94.28</td>
</tr>
</tbody>
</table>

Assume we need the data reported in the first subtable above in a delimited file format in order to import it to other applications, like spreadsheets or databases. Consider the following code:

```plaintext
file out2 / 'out2.csv' /;
```
Note that we set the file attribute page control (.pc) to 5. This means that non-numeric output items will be quoted and each output item will be delimited with commas. Observe that field widths, alignments and horizontal cursor relocations were completely avoided. The put file out2.put will contain the following lines:

```
"capacity (metric tons)","ahmsa","fundidora","sicartsa","hylsa","hylsap"
"blast furnaces",3.25,1.40,1.10,0.00,0.00
"open hearth furnaces",1.50,0.85,0.00,0.00,0.00
"basic oxygen converters",2.07,1.50,1.30,0.00,0.00
"direct reduction units",0.00,0.00,0.00,0.98,1.00
"electric arc furnaces",0.00,0.00,0.00,1.13,0.56
```

While a comma is the most commonly used delimiting character, other delimiters like a blank space (.pc=4) and a tab (.pc=6) may also be used.

For other examples of code that uses the put writing facility, see the various models in the GAMS Model Library.

### 4.43 Solver Usage

For the novice GAMS user, solver usage can be very simple: one runs the model and inspects the listing file to see what the solution is. No knowledge of solver options or solver specific return codes is required. While this is enough for some users, most will quickly find they need some basic knowledge of how to control the solver and interpret the results. Section Controlling a Solver via GAMS Options describes how to set the GAMS options that control a solver. Further, most solvers allow the user to set additional, solver-specific options. These can be set via a solver specific options file, which will be discussed in Section The Solver Options File. However, use of generic GAMS options should be preferred, since a GAMS option setting applies to all solvers and is interpreted by the solvers in a consistent way.

A number of solvers can make use of an initialization of variable and equation values. This will be discussed in Starting Point and Initial Basis.

Further solver specific topics, which are more interesting for advanced users, are discussed in the Sections Solve trace and Branch-and-Cut-and-Heuristic Facility (BCH).

For some hints on how to select a solver, see Choosing an appropriate Solver.
4.43 Solver Usage

4.43.1 Controlling a Solver via GAMS Options

GAMS options can be set on the GAMS command line, e.g.,

$ gams transport iterlim = 100

Additionally, they can be set by an option statement within a GAMS model, e.g.,

option iterlim = 100;

Finally, a model attribute can set a GAMS option for an individual model:

mymodel.iterlim = 100;

The model suffix takes precedence over the option statement, which takes precedence over the command line parameters. If none of these methods is used to set an option, default values apply.

Further, one can unset any model-specific option by assigning it the value \texttt{NA}:

mymodel.iterlim = NA;

Unfortunately, not every option can be set as command line parameter, option statement, and model attribute. We refer to

- Solver-Related Options for the list of solve-related options that can be set via the command line,
- Options that Control Solver-Specific Parameters and Options that Control the Choice of Solver for the list of solve-related options that can be set via the option statement, and
- Model Attributes Mainly Used Before Solve for the list of solve-related model attributes.

4.43.2 The Solver Options File

To specify solver-specific options, it is necessary to use a solver option file. Two things are required to do this: one must create an option file having a proper name, and one must tell the solver to read and use this option file.

To tell a solver to use an option file, one can set the \texttt{optfile} model attribute or the \texttt{optfile} option to a positive value. For example,

model mymodel /all/;
mymodel.optfile = 1;
solve mymodel using nlp maximizing dollars;

The option file takes its name from the solver being used: \texttt{solvername.XYZ}, where \texttt{solvername} is the name of the solver that is specified, and the suffix \texttt{XYZ} depends on the value to which \texttt{optfile} has been set. If its value is 1, the suffix is \texttt{opt}. For example, the option file when calling CONOPT would be called \texttt{conopt.opt}. See the documentation on \texttt{optfile} for more information.

The format of the options file can change marginally from solver to solver. The following illustrates some frequent features of the option file format. However, solvers may vary from this format. Thus, the solver-specific documentation should be checked before using an option file.
• Blank lines in an option file are ignored.
• A comment line might begin with an asterisk (\*) in the first column, is not interpreted by either GAMS or the solver, and is used purely for documentation.
• Each non-comment line contains only one option specification.
• The format for specifying options is as follows:

  keyword(s) [modifier] [value]

The keyword may consist of one or more words and is not case sensitive. The value might be an integer, a real, or a string. Real numbers can be expressed in scientific format, e.g., 1e-6. Note that not all options require modifiers or values.
• Any errors in the spelling of keyword(s) or modifiers will lead to that option being misunderstood and therefore ignored. Errors in the value of an option can result in unpredictable behavior. When detected, errors are either ignored or pushed to a default or limiting value, but not all can or will be detected.

Consider the following CPLEX options file,

* CPLEX options file
  barrier
  crossover 2

The first line begins with an asterisk and therefore contains comments. The first option specifies the use of the barrier algorithm to solve the linear programming problem, while the second option specifies that the crossover option 2 is to be used. Details of these options can be found in Summary of CPLEX Options.

Options can also be defined by generating an options file within the GAMS source code. Consider the following code fragment for using MINOS options file,

```gams
Model m /all/;
Option NLP = MINOS;
* MINOS options file
$onecho > minos.opt
  scale option 2
  completion partial
$offecho
m.OptFile = 1;
Solve m maximizing z using nlp ;
```

The first option sets the scale option to a value of 2. In this case, the keyword 'scale option' consists of two words. In the second line, the 'completion' option is set to 'partial'. Details of these options can be found in Summary of MINOS Options.

### 4.43.2.1 Dot Options

Dot options in a solver option file allow users to associate values to variables and equations using the GAMS name of the variables and equations. The general syntax of a dot option in the option file is as follows:

```
(variable/equation name).optionname (value)
```
Dot options can be specified for **all**, a **block**, a **slice**, and a **single** variable and equation. Please note that a specific dot option may only apply to variables or equations (e.g. the GAMS/Gurobi dot option **prior** applies to variables only). The following example makes the use of the dot option clear.

For example, suppose one has a GAMS declaration:

```gams
Set i /i1*i5/;
Set j /j2*j4/;
Variable v(i,j);
Equation e(i,j);
```

Consider the following lines in an option file with the imaginary option name **dotopt**:

<table>
<thead>
<tr>
<th>Line in option file</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables.dotopt 1</td>
<td>Sets the value of all variables to 1</td>
</tr>
<tr>
<td>equations.dotopt 2</td>
<td>Sets the value of all equations to 2</td>
</tr>
<tr>
<td>v.dotopt 3</td>
<td>Sets the value of the variables in block v to 3</td>
</tr>
<tr>
<td>e.dotopt(∗,∗) 4</td>
<td>Sets the value of the equations in block e to 4</td>
</tr>
<tr>
<td>v.dotopt(∗,'j2') 5</td>
<td>Sets the value of the variables v that have j2 in the second index position (slice) to 5</td>
</tr>
<tr>
<td>e.dotopt('i3',∗) 6</td>
<td>Sets the value of the equations e that have i3 in the first index position (slice) to 6</td>
</tr>
<tr>
<td>w.dotopt('i2') 7</td>
<td>Sets the value of the single variables v('i2') to 7</td>
</tr>
<tr>
<td>e.dotopt('i3','j3') 8</td>
<td>Sets the value of the single equations e('i3','i3') to 8</td>
</tr>
</tbody>
</table>

The values of the dot option are applied in correspondence to the sequence in which they appear in the option file. In the current example, the values of **dotopt** for the equation **e** would be as follows:

<table>
<thead>
<tr>
<th>e.dotopt</th>
<th>i1</th>
<th>i2</th>
<th>i3</th>
</tr>
</thead>
<tbody>
<tr>
<td>j2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>j3</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>j4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

### 4.43.3 Starting Point and Initial Basis

#### 4.43.3.1 Starting Point

NLP solvers that search for a locally optimal solution of a NLP require an initial point to start their search. Further, the closer this initial point is to a local optimum, the less effort the solver may have to spend. The latter can also be true for solvers that search for global optimal solutions, such as most LP or MIP solver or global MINLP solvers.

Because of this immense importance of a starting point, GAMS always passes a starting point to a solver. By default, the point passed on by GAMS is given by the level and marginal attributes of variables and equations. If these values have not been set yet, default values are used. This default value is zero, except for variables whose bounds would forbid this value. In this case, the bound closest to zero is used.

In addition to setting these values explicitly in a GAMS model, a user can also load them from a **save file** or a GDx point file via **execute_loadpoint**. The latter may have been generated by running a related model and using option **savepoint**. Further, in models with several solve statements, the solution from...
one solve, if any, is used to initialize the starting point for a succeeding solve. This happens automatically since solutions from a solve statements are also stored in the level and marginal values of variables and equations. Finally, note that model attribute defpoint can be used to force sending the default starting point to a solver.

For some solvers, in particular for MIP or MINLP, an option may have to be set to make use of the starting point. Further, some solvers offer the possibility to make use of a partial starting point or use the starting point as a guide for the search. For details, see the specific solver manuals and look for parameters like mipstart and the use of the GAMS parameter tryint.

### 4.43.3.2 Initial Basis

While for some solvers, the values of a starting point are sufficient to initialize the search, active-set algorithms make use of a different form of starting information. An active-set algorithm tries to identify which constraints are active (or binding) in a feasible or optimal solution, that is, which variables are at one of their bounds and for which equations the activity equals the right-hand-side and the marginal is non-zero. For example, the simplex method for linear programs is an active-set algorithm. The classification of constraints as either active or inactive is closely linked to a basis. Active constraints are called nonbasic and inactive constraints are called basic. A basis that specifies the active and inactive constraints in an optimal solution is called an optimal basis.

Active-set algorithms may start by guessing an initial basis and then iteratively updating this basis until an optimal basis is found. Solution time may be reduced substantially if a good approximation of the optimal basis can be identified and passed on to the solver. Such a user-provided initial basis is called an advanced basis. However, provision of an advanced basis does not always help. For example, the presolving algorithms used in some solvers may cause the advanced basis to be ignored. Further, solvers may perform poorly when the advanced basis provided is worse than what the solver would otherwise have constructed with its own heuristics. Finally, not all algorithms benefit from the provision of an advanced basis: interior-point algorithms are a notable example.

GAMS provides a hint to the solver to suggest whether a basis can or should be extracted from the initial point. The hint is based on the number of rows (aka single equations) with nonzero marginals and the value of the bratio option and is computed internally as:

\[
\text{hint} := (0 == \text{bRatio}) \text{ or } (\text{rowsWithNonzeroMarg} > \text{nRows} \times \text{bratio})
\]

While the setting of the variable level and marginal values is important in specifying an advanced basis, only the marginal values for the rows are relevant in computing the basis hint and in choosing a value for bratio. For example, in a problem with 1000 rows and with the default value of bratio (0.25), GAMS would suggest a basis only if at least 250 nonbasic rows exist.

Note that the default starting point - all marginals zero - is usually not sufficient for the construction of an initial basis. If the automatic transfer of a basis from one solve statement to the next leads to poor behavior in the solver, setting the option bratio to 1 will cause the basis hint to be false and suppress the use of an initial basis. Setting the model attribute defpoint to 1 or 2 will achieve the same result. Conversely, setting bratio to 0 will force the basis hint to be true.

Active-set solvers in GAMS typically use the basis hint value to decide whether to extract (and use) the advanced basis available in the starting point. Details can vary by solver, but typically it works as follows:

- Variables with a zero level value are suggested to be nonbasic if and only if they have a non-zero marginal value.
- Variables with a non-zero level value are suggested to be nonbasic if the level value is equal one of the variable bounds. The marginal values may also be used here: nonbasic variables correspond with non-zero marginal values.
• Equations with non-zero marginal are suggested to be nonbasic. Otherwise, they are suggested to be basic.

Thus, a user can attempt to explicitly provide an initial basis by setting a corresponding starting point. That is, one can set a guess for an initial basis by specifying

• non-zero marginals for equations that are predicted to be active in a solution
• non-zero marginals for variables that are predicted to be at their bound in a solution
• non-zero levels for the variables that are predicted to be non-zero in a solution

4.43.4 Trace Features

Sometimes it might be useful to get certain information about a particular solve statement (or GAMS job) in a compact and customizable format.

• The Trace file facility allows to create files containing information about the "end data" of a GAMS job and the contained solve statements.
• The Solve trace facility can provide more detailed information about the solution progress of a particular solve statement (e.g. objective value of the incumbent solution or the best dual bound every five seconds)

4.43.4.1 Trace File

The trace file contains information about the "end data" of a GAMS job and the contained solve statements. Creation of a trace file can be activated via command line parameter trace (see example below). The trace feature supports several predefined formatting options, that is differently formatted trace files can be created, depending on what output information is desired. The trace file format option can be set via command line parameter traceOpt. The trace file header defines the contained trace records and the associated trace record fields.

Note

• Trace information is appended to existing trace files in the format of the existing trace file. That is, if a previous trace file of the same name already exists, then all new trace data will be appended in the initial format, no matter if the current traceOpt value actually implies a different format.
• Trace file headers can be modified, i.e. the predefined formats can be customized and trace record fields can be added or removed as needed.

4.43.4.1.1 Trace Records A trace file can contain different types of trace records:
<table>
<thead>
<tr>
<th>Trace Record Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GamsStep</td>
<td>A GamsStep refers to a GAMS execution phase (there are potentially multiple phases of GAMS execution in a single GAMS Job). For example, a trace file created with traceopt=0 for the [TRNSPORT] model, which contains a single solve statement, will contain two GamsSteps, one referring to the GAMS execution phase before the solve statement, the other one referring to the execution phase after the solve statement. Note that this may change depending on the setting of solveLink.</td>
</tr>
<tr>
<td>GamsExit</td>
<td>GamsExit describes the final GamsStep.</td>
</tr>
<tr>
<td>GamsSolve</td>
<td>GamsSolve describes a trace record referring to a solve statement. If a file has multiple solve statements, the corresponding trace file can have multiple GamsSolve trace records.</td>
</tr>
</tbody>
</table>

### 4.43.4.1.2 Trace Record Fields

For every trace record, the trace file can contain multiple fields from the following list.

<table>
<thead>
<tr>
<th>Name of Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS</td>
<td>CNS solver.</td>
</tr>
<tr>
<td>ComputerName</td>
<td>Computer name.</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of optimization: 0=min, 1=max.</td>
</tr>
<tr>
<td>DNLP</td>
<td>DNLP solver.</td>
</tr>
<tr>
<td>EMP</td>
<td>EMP solver.</td>
</tr>
<tr>
<td>ETAlg</td>
<td>Elapsed time it took to execute the solve algorithm (see corresponding model attribute etAlg).</td>
</tr>
<tr>
<td>ETSolve</td>
<td>Elapsed time it took to execute a solve statement in total (see corresponding model attribute etSolve).</td>
</tr>
<tr>
<td>ETSolver</td>
<td>Elapsed time taken by the solver only (see corresponding model attribute etSolver).</td>
</tr>
<tr>
<td>GamsCloseDownTime</td>
<td>The time it takes to close down GAMS (which partly depends on command line parameters) including the time to write the file summary, a GDX file, a (obfuscated) save file, a parameter file.</td>
</tr>
<tr>
<td>GamsCompTime</td>
<td>Time of the GAMS compilation phase.</td>
</tr>
<tr>
<td>GamsElapsedTime</td>
<td>Elapsed time since the start of a GAMS run in seconds.</td>
</tr>
<tr>
<td>GamsElements</td>
<td>Number of labels.</td>
</tr>
<tr>
<td>GamsErrorCount</td>
<td>Number of compilation and execution errors.</td>
</tr>
<tr>
<td>GamsExecTime</td>
<td>GAMS execution time.</td>
</tr>
<tr>
<td>GamsLineNumber</td>
<td>Number of lines. This corresponds to the number of lines in the echo print of the input file.</td>
</tr>
<tr>
<td>GamsReturnCode</td>
<td>GAMS return code.</td>
</tr>
<tr>
<td>GamsStartupTime</td>
<td>The time it takes to start up GAMS including the time to check and dump command line parameters, initialize certain internal data structures, loading a work file, reading a license file.</td>
</tr>
<tr>
<td>GamsSymbols</td>
<td>Number of symbols (also called identifiers) including names of intrinsic functions and predefined symbols.</td>
</tr>
<tr>
<td>GamsTotalTime</td>
<td>Total time of the corresponding trace record. Does only apply for trace records of type GamsStep and GamsExit.</td>
</tr>
<tr>
<td>GamsVersionID</td>
<td>String that contains a GAMS version ID, e.g. WEX282-282 for the Windows 64bit version of GAMS 28.2.</td>
</tr>
<tr>
<td>InputFileName</td>
<td>GAMS model filename.</td>
</tr>
<tr>
<td>JobDate</td>
<td>Day when the job started.</td>
</tr>
<tr>
<td>Name of Field</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JobTime</td>
<td>Time when the Job started.</td>
</tr>
<tr>
<td>JulianDate</td>
<td>Julian date number with start day/time of job.</td>
</tr>
<tr>
<td>LP</td>
<td>LP solver.</td>
</tr>
<tr>
<td>Marginals</td>
<td>Indicates availability of marginals: 0=no, 1=yes.</td>
</tr>
<tr>
<td>MCP</td>
<td>MCP solver.</td>
</tr>
<tr>
<td>MINLP</td>
<td>MINLP solver.</td>
</tr>
<tr>
<td>MIP</td>
<td>MIP solver.</td>
</tr>
<tr>
<td>MIQCP</td>
<td>MIQCP solver.</td>
</tr>
<tr>
<td>ModelGenerationTime</td>
<td>Model generation time. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>modelName</td>
<td>Model name. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>ModelStatus</td>
<td>Model status. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>ModelType</td>
<td>Model type. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>MPEC</td>
<td>MPEC solver.</td>
</tr>
<tr>
<td>NLP</td>
<td>NLP solve.</td>
</tr>
<tr>
<td>NumberOfDiscreteVariables</td>
<td>Number of discrete variables. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>NumberOfDomainViolations</td>
<td>Number of domain violations (see also model attribute domUsd).</td>
</tr>
<tr>
<td>NumberOfEquations</td>
<td>Number of equations. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>NumberOfInstructions</td>
<td>Length of non-linear code.</td>
</tr>
<tr>
<td>NumberOfIterations</td>
<td>Number of iterations. Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>NumberOfNodes</td>
<td>Number of nodes. Refers to the solve of the last model in the part of the program that corresponds to the trace record. Only relevant if some tree search algorithm (e.g. branch-and-bound was used).</td>
</tr>
<tr>
<td>NumberOfNonlinearNonZeros</td>
<td>Number of non-zeroes. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>NumberOfNonZeros</td>
<td>Number of variables. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>NumberOfVariables</td>
<td>Number of variables. Refers to the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>ObjectiveValue</td>
<td>Objective value. Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>ObjectiveValueEstimate</td>
<td>Estimate of the best possible solution for a mixed-integer model (aka best bound). Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>OptionFile</td>
<td>Solver option file number. Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>Platform</td>
<td>Platform ID, e.g. WEX for the Windows 64bit.</td>
</tr>
<tr>
<td>QCP</td>
<td>QCP solver.</td>
</tr>
<tr>
<td>RMINLP</td>
<td>RMINLP solver.</td>
</tr>
<tr>
<td>RMIP</td>
<td>RMIP solver.</td>
</tr>
<tr>
<td>RMIQCP</td>
<td>RMIQCP solver.</td>
</tr>
<tr>
<td>RMPEC</td>
<td>RMPEC solver.</td>
</tr>
<tr>
<td>SolveLine</td>
<td>Line number of the last solve statement in the part of the program that corresponds to the trace record. This corresponds to the line number in the echo print of the input file.</td>
</tr>
<tr>
<td>Name of Field</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SolveNumber</td>
<td>Number of solve statement for a particular model. For example, solve myModelA...; solve myModelB...; solve myModelA...; results in solve numbers 1, 1, 2.</td>
</tr>
<tr>
<td>SolverCalcTime</td>
<td>Time spent in function and derivative calculations (deprecated, see also model attribute resCalc).</td>
</tr>
<tr>
<td>SolverElapsedTime</td>
<td>Elapsed time taken by the solver when solveLink=0 is used (almost identical to model attribute etSolver).</td>
</tr>
<tr>
<td>SolverID</td>
<td>Solver ID number.</td>
</tr>
<tr>
<td>SolverName</td>
<td>Name of the solver. Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>SolverReadTime</td>
<td>Time to import model (deprecated, see also model attribute resIn).</td>
</tr>
<tr>
<td>SolverRealTime</td>
<td>Elapsed time taken by the solver when solveLink=0 is used (almost identical to model attribute etSolver).</td>
</tr>
<tr>
<td>SolverSignature</td>
<td>String containing a solver signature, e.g. IBM ILOG CPLEX 28.2.0 r750fa45 Released Aug 19, 2019 WEI x86 64bit/MS Window. Refers to the solver used to solve the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>SolverStatus</td>
<td>Solver Status. Refers to the solve of the last model in the part of the program that corresponds to the trace record.</td>
</tr>
<tr>
<td>SolverTime</td>
<td>Time used to solve the model in seconds reported by the solver (see also model attribute resUsd).</td>
</tr>
<tr>
<td>SolverVersion</td>
<td>Solver Version (see also model attribute sysVer).</td>
</tr>
<tr>
<td>SolverWriteTime</td>
<td>Time to export solution (deprecated, see also model attribute resOut).</td>
</tr>
<tr>
<td>User1</td>
<td>Content of user1 string.</td>
</tr>
<tr>
<td>User2</td>
<td>Content of user2 string.</td>
</tr>
<tr>
<td>User3</td>
<td>Content of user3 string.</td>
</tr>
<tr>
<td>User4</td>
<td>Content of user4 string.</td>
</tr>
<tr>
<td>User5</td>
<td>Content of user5 string.</td>
</tr>
<tr>
<td>UserName</td>
<td>User name.</td>
</tr>
</tbody>
</table>

### 4.43.4.1.3 Trace Report

GAMS can create trace reports from trace files, if started with command line parameter action=gt. The following sequence of commands loads model [TRNSPORT] from the GAMS model library, runs trnsport.gms and creates a tracefile trc.txt, and produces a trace report plus a trace summary from the trace file.

```
$ gamslib trnsport
$ gams trnsport trace=trc.txt
$ gams trc.txt a=gt
```

The trace file trc.txt contains the default trace record fields for the GamsStep and GamsSolve trace records:

- Trace Record Definition
- GamsStep
  - JobDate JobTime InputFileName GAMS GamsVersionID GamsReturnCode GamsErrorCount GamsStartupTime GamsCompTime GamsExecTime GamsCloseDownTime GamsTotalTime
- GamsSolve
  - JobDate JobTime InputFileName ModelType SolverName SolverStatus ModelStatus SolveNumber SolveLine SolveNumber SolveLine
  - NumberOfIterations ModelGenerationTime SolverReadTime SolverCalcTime SolverWriteTime ObjectiveValue
The trace summary trc.sum contains condensed information about the trace records:

Trace Summary with TL=0 for [...]\trc.txt
Trace Records = 3, Error Records = 0, First Date = 09/25/19 07:04:00

The trace report is created in trc.lst and contains detailed information about the trace records from the underlying trace file including a ModelSolveStat matrix that assigns a score from 0 to 9 to all potential combinations of model status and solver status:

[...]
Trace Report for File: [...]\trc.txt

Using ModelSolveStat for TL=0

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>2 9 9 9 9 9 9 . . 9 . . . . .</td>
</tr>
<tr>
<td>3 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>4 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>5 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>6 . 5 5 5 5 . . 5 . . . . . .</td>
</tr>
<tr>
<td>7 9 9 9 9 9 . . 9 . . . . . .</td>
</tr>
<tr>
<td>8 9 9 9 9 9 . . 9 . . . . . .</td>
</tr>
<tr>
<td>9 . 5 5 5 5 . . 5 . . . . . .</td>
</tr>
<tr>
<td>10 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>11 . . . . . . 5 . . . . . . .</td>
</tr>
<tr>
<td>12 . . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>13 . . . . . . 3 3 3 . . . . .</td>
</tr>
<tr>
<td>14 4 4 4 4 4 6 . 4 . . . . . .</td>
</tr>
<tr>
<td>15 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>16 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>17 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>18 9 . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>19 9 . . . . . . . . . . . . .</td>
</tr>
</tbody>
</table>

Marked trace records are listed below:

Record Date Time Filename Type Solver Message
* Trace Record Definition
* GamsStep
* JobDate JobTime InputFileName GAMS GamsVersionID GamsReturnCode GamsErrorCount GamsStartupTime GamsCompTime GamsExecTime GamsCloseDownTime GamsTotalTime "User1"
* GamsSolve
* JobDate JobTime InputFileName ModelType SolverName SolverStatus ModelStatus SolveNumber SolveLine ModelGenerationTime SolverReadTime SolverCalcTime SolverWriteTime Objective Value "User1"
* NumberOfIterations ModelGenerationTime SolverReadTime SolverCalcTime SolverWriteTime Objective Value "User1"

First time stamp = 09/25/19 07:04:00
Last time stamp = 09/25/19 07:04:00
Total trace records = 3
Total comment records = 7

Total GAMS records = 2 all return codes
   1 RC= 0 Normal completion
   1 RC= 1 Executing subsystem

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>total</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>compile</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>execute</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>closedown</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>total</td>
<td>0.03</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Total SOLVE records = 1 all return codes
   solvestatus 1 RC= 1 1 Normal Completion
   modelstatus 1 RC= 1 1 Optimal

<table>
<thead>
<tr>
<th>Numbers</th>
<th>total</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>equations</td>
<td>6</td>
<td>6.00</td>
</tr>
<tr>
<td>variables</td>
<td>7</td>
<td>7.00</td>
</tr>
<tr>
<td>nonzeros</td>
<td>19</td>
<td>19.00</td>
</tr>
<tr>
<td>nonlinear</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>instruction</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>equations</td>
<td>6</td>
<td>6.00</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>total</td>
<td>mean</td>
</tr>
<tr>
<td>generation</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>calculation</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>output</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>total solve</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

SOLVESTAT Cross Tabulation

    1  2  3  4  5  6  7  8  9  10 11 12 13  tot
      LP  1  .  .  .  .  .  .  .  .  .  .  .  1
    total  1  0  0  0  0  0  0  0  0  0  0  0  1

MODELSTAT Cross Tabulation

    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
      LP  1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    total  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

GTRACE TIME = 0.000 SECONDS VERID WEX282-282 [...]

The trace report may be particularly interesting, if the underlying trace file contains multiple trace records from different solve statements. In combination with command line parameter traceLevel, it is easy to
filter for GamsSolve trace records with a low ModelSolveStat score, i.e. undesirable combinations of model status and solver status. If we run [TRNSPORT] again with an iteration limit of 0 (remember that the trace information will be appended to the trace file from the example above) and recreate the trace report as follows

```bash
$ gams trnsport trace=trc.txt iterlim=0
$ gams trc.txt a=gt
```

the trace report shows two solve records and the obtained statuses:

```
Total SOLVE records = 2 all return codes

  solvestatus           1 RC=  1 1 Normal Completion
  solvestatus           1 RC=  2 2 Iteration Interrupt
  modelstatus           1 RC=  1 1 Optimal
  modelstatus           1 RC=  6 6 Intermediate Infeasible
```

A look at the ModelSolveStat matrix from the initial example, tells us that a combination of model status 6 (Intermediate Infeasible) in combination with solve status 2 (Iteration Interrupt) results in a score of 5. If we increase the traceLevel to a value greater than or equal to 5 and recreate the trace report again, e.g. via

```bash
$ gams trc.txt a=gt tracelevel=5
```

the ModelSolveStat matrix at the beginning of the trace report changes such that only the scores greater than 5 for accepted combinations of model and solver status are shown

Using ModelSolveStat for TL=5

```
  1  2  3  4  5  6  7  8  9 10 11 12 13
1  9 . . . . . . . . . . . .
2  9  9  9  9  9 .  9 . . .
3  9 . . . . . . . . . . .
4  9 . . . . . . . . . . .
5  9 . . . . . . . . . . .
6 . . . . . . . . . . . .
7  9  9  9  9  9 .  9 . . .
8  9  9  9  9  9 .  9 . . .
9 . . . . . . . . . . . .
10 9 . . . . . . . . . . .
11 . . . . . . . . . . . .
12 . . . . . . . . . . . .
13 . . . . .  6 . . . . .
14 . . . . . . . . . . . .
15 9 . . . . . . . . . . .
16 9 . . . . . . . . . . .
17 9 . . . . . . . . . . .
18 9 . . . . . . . . . . .
19 9 . . . . . . . . . . .
```

and GAMS throws an execution error

```text
--- Starting trace report generation
*** Status: Execution error(s)
```

The trace file facility in combination with the trace report can be particularly useful for automated quality assurance tests where certain expected outcomes should be audited.
4.43.4.2 Solve trace

In order to do accurate performance evaluations it may be useful to obtain more detailed information about a solve than the “end data” that the trace file provides. E.g., for a branch-and-bound based solver, one may want to have intermediate information about the values of primal and dual bounds at the root node and subsequent nodes within the search.

The solve trace option that is implemented in some of the GAMS solver interfaces allows users to output solve information, e.g., primal and dual bounds, for every n nodes or at every time step. For example, the user may be interested in the objective value of the incumbent solution or the best dual bound on the optimal value every 50 nodes and every five seconds of the solve.

Note

The solve trace file format and options may change in a future GAMS release.

The solve trace option is invoked via a GAMS solver options file. Usually, options to specify a filename of the trace file to be created and options to specify time and node intervals are available. Please refer to the GAMS solver manuals for the exact names of these options (search for solvetrace or miptrace).

The solve trace file is written in comma-separated-value (CSV) format, where the entries in each line have the following meaning:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineNum</td>
<td>a line index</td>
</tr>
<tr>
<td>seriesID</td>
<td>indicator why the line was written: S = start of search, N = node frequency, T = time frequency, E = end of search</td>
</tr>
<tr>
<td>node</td>
<td>number of enumerated branch-and-bound nodes</td>
</tr>
<tr>
<td>seconds</td>
<td>time since the solving started</td>
</tr>
<tr>
<td>bestFound</td>
<td>primal bound, i.e., objective value of incumbent solution</td>
</tr>
<tr>
<td>bestBound</td>
<td>dual bound, i.e., bound on optimal value</td>
</tr>
</tbody>
</table>

A sample solve trace file including statistics of a GAMS run using the MIP model [DICE] and the solver XPRESS looks as follows:

* miptrace file miptrace.mtr: ID = XPRESS.1 Instance = dice
* fields are lineNum, seriesID, node, seconds, bestFound, bestBound
1, S, 1, 0.078, na, na
2, N, 101, 0.781, 21, 23.9369
3, N, 201, 0.875, 21, 23.5564
4, N, 304, 1.031, 21, 23.5564
5, E, 399, 1.094, 21, 21
* miptrace file miptrace.mtr closed

See also the slides for the presentation Advanced Use of GAMS Solver Links (2013) and the accompanying scripts for some ideas on what to do with the solve trace functionality.

4.43.5 Branch-and-Cut-and-Heuristic Facility (BCH)

Global search algorithms can sometimes significantly benefit from user supplied routines that support the solution process of an hard optimization problem. For example, branch-and-cut solvers (e.g., CPLEX,
Gurobi, SCIP, Xpress) can profit from user-supplied cutting planes or good feasible solutions. GAMS users could supply these as part of the model given to the solver, by adding a set of constraints representing likely to be violated cuts and an initial solution (possibly in combination with GAMS parameters like tryint and solver-specific options like mipstart in CPLEX). However, this does not allow a dynamic interaction between a running solver and user supplied routines that, for example, use a current relaxation solution to construct cutting planes or feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) facility attempts to automate all major steps necessary to make callbacks that certain solvers provide for such usage available to the GAMS user. This allows GAMS users to apply complex solution strategies without having to have intimate knowledge about the inner workings of a specific solver.

Currently, only two solvers support the BCH facility: CPLEX and SBB. With GAMS/CPLEX, user supplied GAMS programs that implement primal heuristics and cut generation can be used. With SBB, only primal heuristics are possible.

As the name indicates, the BCH facility has been designed with the solving process of a branch-and-cut solver (e.g., CPLEX, Gurobi, SCIP, Xpress) in mind. Such solvers often allow to call a user supplied routine after a node in the branch-and-bound (B&B) tree has been processed. Within that routine, available information like the solution of a relaxation (often an LP or NLP) at that node and the current incumbent, if any, is exported by the BCH facility into a GDX file using the original GAMS namespace. Next, different user supplied GAMS programs can be called, e.g., for finding cuts which are violated by the relaxation solution (cut generator) or to find new incumbents (primal heuristic). These GAMS programs should import the information from the GDX file and do their computations. After termination, the BCH facility resumes control, reads the findings from the GAMS program and passes them to the solver.

A relaxation solution may be exported into a file `bchout.gdx` by the BCH facility. This GDX file does not only contain the variable values as level values (.l), but also variable bounds (.lo and .up). For a B&B solver, these are the local bounds at this node. Hence, they reflect branching decisions made in the B&B tree and bound tightenings that were deduced by the solver. In a similar way, the BCH facility may export an incumbent solution to the GDX file `bchout_i.gdx`. The bounds for the incumbent solution reflect global bounds, i.e., the original bounds, possibly tightened by the solver. GDX files can be imported by the GAMS program using the compile time `$load` or run time `execute_load`.

The BCH facility is activated and controlled by setting certain options in the solvers options file. The precise names and meanings of the options may vary from one solver to another. Therefore, also the corresponding GAMS solver manual should be checked. The options that come with the BCH facility can be used to define the calls of the users GAMS programs, to determine when they should be called, and to overwrite the filenames for the GDX files (to avoid name clashes). General BCH related options are the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserGDXIn</td>
<td>The name of the GDX file read back into the solver.</td>
<td>bchin.gdx</td>
</tr>
<tr>
<td>UserGDXName</td>
<td>The name of the GDX file exported from the solver with the solution at the node.</td>
<td>bchout.gdx</td>
</tr>
<tr>
<td>UserGDXNameInc</td>
<td>The name of the GDX file exported from the solver with the incumbent solution.</td>
<td>bchout_i.gdx</td>
</tr>
<tr>
<td>UserGDXPrefix</td>
<td>Prefix to add to UserGDXIn, UserGDXName, and UserGDXNameInc</td>
<td>empty</td>
</tr>
<tr>
<td>UserJobID</td>
<td>Postfix to add to listing and log filenames and to UserGDXIn, UserGDXName, and UserGDXNameInc. Further, --UserJobID is added to calls to users GAMS programs.</td>
<td>empty</td>
</tr>
<tr>
<td>UserKeep</td>
<td>Calls users GAMS programs with gamskeep instead of gams. The use of gamskeep will preserve the scratch directory similar to the command line option keep</td>
<td>0</td>
</tr>
</tbody>
</table>

In the following, the interface for the available callbacks are explained in more detail and corresponding options are listed.
4.43.5.1 Primal Heuristics

In the primal heuristic callback, the user can provide a GAMS program which tries to construct a feasible solution based on the information provided by the solver, e.g., a current relaxation solution and the current incumbent. Thus, the GAMS program could attempt to repair infeasibilities in the relaxation solution or try to improve the incumbent from the solver.

If the GAMS program finds a new solution, it should store it in the level values of variables that correspond to the original variables. For example, if the original model uses binary variable \( \text{open}(i,t) \), then at the end of the GAMS program \( \text{open}.l(i,t) \) should contain a 0 (zero) or a 1 (one). The BCH facility calls the GAMS program and instructs GAMS to store the results in a GDX file at termination. This GDX file is then read in again by the BCH facility and the solution is passed back to the solver. The solver checks this solution for infeasibilities and in case this check is passed and the solution is better than the best known solution, the solver updates it's incumbent.

If the GAMS program cannot find a feasible solution, it can terminate with an execution error triggered by an \texttt{abort} statement to prevent the BCH facility from reading the results from the heuristic run.

BCH parameters to control the primal heuristic call are typically the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserHeurFreq</td>
<td>Determines the frequency of the heuristic call.</td>
<td>10</td>
</tr>
<tr>
<td>UserHeurMult</td>
<td>Determines the multiplier for the frequency of the heuristic call.</td>
<td>2</td>
</tr>
<tr>
<td>UserHeurInterval</td>
<td>Determines the interval when to apply the multiplier for the frequency of the heuristic call. For example, for the first 100 (UserHeurInterval) nodes, the solver calls the heuristic every 10th (UserHeurFreq) node. After 100 nodes, the frequency gets multiplied by 10 (UserHeurMult), so that for the next 100 nodes the solver calls the heuristic every 20th node. For nodes 200-300, the heuristic get called every 40th node, for nodes 300-400 every 80th node and after node 400 every 100th node.</td>
<td>100</td>
</tr>
<tr>
<td>UserHeurFirst</td>
<td>For how many of the first nodes the heuristic should be called.</td>
<td>10</td>
</tr>
<tr>
<td>UserHeurObjFirst</td>
<td>Similar to UserHeurFirst, but specifies for how many of the first nodes the heuristic should be called if the optimal value of the current nodes relaxation promises a significant improvement of the current incumbent, i.e., the optimal value of the relaxation at the node has to be closer to the current dual bound than the current primal bound.</td>
<td>solver dependent</td>
</tr>
<tr>
<td>UserHeurNewInt</td>
<td>Whether to calls the heuristic when the solver found a new feasible solution.</td>
<td>no</td>
</tr>
<tr>
<td>UserHeurCall</td>
<td>Arguments to the GAMS call to invoke the heuristic GAMS program.</td>
<td>empty</td>
</tr>
</tbody>
</table>

As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the primal heuristic in the GAMS program \texttt{bchoil.h.inc} when using GAMS/CPLEX could be

```
userheurcall      bchoil_h.inc mip cplex optcr 0 reslim 10
userheurfirst     5
userheurfreq      20
userheurinterval  1000
```
4.43.5.2 Cutting Planes

In the cut generator callback, the user can provide a GAMS program which tries to find a linear cut (that is, a linear inequality) that is violated by the relaxation solution. The solver would then add these cuts to its cut pool. Typically, it then resolves the relaxation at the node and calls the cut generator again. If no cutting planes are found, the solver will continue, e.g., by processing the next node. Please note that the solver cannot perform validity checks on the provided cuts. Hence, it is possible to cut off areas of the feasible region, including optimal solutions.

Exporting cuts is a little more complicated than a solution because next to the cut coefficients, also the sense and the right-hand-side of the cut inequality needs to be exported. Further, exporting several cuts with one call should be possible. For this purpose, the GAMS program has to define and fill a set cc and parameters `numcuts`, `rhs_c(cc)`, and `sense_c(cc)` appropriately. The set cc is used as a cut index. It can be larger than the number of actually generated cuts.

Note

The elements of the cut index set must form a series of integers starting at 1 (1, 2, 3,...).

Parameter `numcuts` should specify the number of added cuts. `rhs_c(cc)` should store the right-hand-side of each cut. Finally, `sense_c(cc)` should store the sense of each cut, which must be 1 for lower-equal (≤), 2 for equal (=, rather unusual for cuts), and 3 for greater-equal (≥). The corresponding declaration in GAMS code may be

```gams
$set MaxCuts 100
Set cc 'cuts' / 1 * %MaxCuts% /;
Parameters numcuts 'number of cuts to be added' / 0 /
  rhs_c(cc) 'cut rhs'
  sense_c(cc) 'the sense of the cuts';
```

The only thing missing are the cut coefficients. As it should be possible to return more than one cut, using variable attributes like level values is not sufficient. Therefore, for each variable that is part of a cut, a new parameter must be added in the GAMS program. The name of the parameter must be the name of the corresponding variable with an additional `_c` at the end. Further, the parameter must be indexed like the variable, but with the cut index set cc added at the beginning. For example, assume variable `open(i,t)` should be part of a cut. Then the cut coefficients should be stored in a parameter `open_c(cc,i,t)`, e.g.,

```gams
Parameter open_c(cc,i,t) 'coefficients of variable open(i,t) in cut cc';
```

The BCH facility reads all parameters that end in `_c`, takes the base name and looks for a variable with that name and indices and builds up the cut matrix. A cut cannot introduce a new variable into the model. All cuts added to the model are assumed to be global cuts, that is, they need to be valid for the entire problem, not just for the current node.

BCH parameters to control the cut generation call are typically the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserCutFreq</td>
<td>Determines the frequency of the cut generator call.</td>
<td>10</td>
</tr>
<tr>
<td>UserCutMult</td>
<td>Determines the multiplier for the frequency of the cut generator call.</td>
<td>2</td>
</tr>
<tr>
<td>UserCutInterval</td>
<td>Determines the interval when to apply the multiplier for the frequency of the cut generator call. See <code>UserHeurInterval</code> for details.</td>
<td>100</td>
</tr>
<tr>
<td>UserCutFirst</td>
<td>Calls the cut generator for the first n nodes.</td>
<td>10</td>
</tr>
<tr>
<td>UserCutNewInt</td>
<td>Whether to call the cut generator if the solver found a new integer feasible solution.</td>
<td>no</td>
</tr>
<tr>
<td>UserCutCall</td>
<td>Arguments to the GAMS call to invoke the cut generator GAMS program.</td>
<td>empty</td>
</tr>
</tbody>
</table>
As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the cut generator in the GAMS program bchoil_c.inc when using GAMS/CPLEX could be

```
usercutcall  bchoil_c.inc mip cplex
usercutfirst  0
usercutfreq  0
usercutnewint yes
```

### 4.43.5.3 Incumbent Callbacks

The incumbent callbacks can be used to execute a GAMS program when the solver found a new feasible solution that improves the incumbent. Additionally, the incumbent check callback `UserIncbCall` can be used to notify the solver whether the given feasible solution should be accepted by the solver. This allows to implement a filtering mechanism that forces a solver to search for additional solutions even though an optimal solution might have been found already. Furthermore, the callback `UserLazyConCall` allows to add lazy constraints after an incumbent has been found.

The following parameters control the incumbent callbacks:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserIncbCall</td>
<td>Arguments to the GAMS call to invoke the incumbent checking GAMS program. The incumbent is rejected if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.</td>
<td>empty</td>
</tr>
<tr>
<td>UserIncbICall</td>
<td>Arguments to the GAMS call to invoke the incumbent reporting GAMS program.</td>
<td>empty</td>
</tr>
<tr>
<td>UserLazyConCall</td>
<td>Arguments to the GAMS call to invoke the lazy constraint adding GAMS program. Lazy constraints that cut off the incumbent are expected (in a format similar to UserCutCall) if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.</td>
<td>empty</td>
</tr>
</tbody>
</table>

### 4.43.5.4 Examples

The GAMS model library contains a few examples to show to use the BCH facility:

- **bchtlbas.gms**: Trim Loss Minimization with Heuristic using BCH Facility This model implements a very simple LP/MIP based primal heuristic for the trimloss minimization problem.

- **bchfcnet.gms**: Fixed Cost Network Flow Problem with Cuts using BCH Facility This model implements simple but difficult to separate cuts for a network design problem. The global solver BARON is used to find violated cuts by solving a non-convex MINLP.

- **bchtmknap.gms**: Multi knapsack problem using BCH Facility This model implements simple cover inequalities for the multi-knapsack problem.

- **bchoil.gms**: Oil Pipeline Design Problem using BCH Facility This is the most complex example. It implements three different primal heuristics: an initial heuristic based on a simplified cost structure, a rounding heuristic, and a local branching heuristic. In addition, complex cuts are generated by solving regionalized versions of the original problem.
4.43 Solver Usage

- **dicegrid.gms**: MIP Decomposition and Parallel Grid Submission - DICE Example
  This example uses many of the UserJobID option to rename files, since running multiple jobs in parallel requires the use of different filenames. This example also uses the incumbent reporting call UserIncbCall.

- **solnpool.gms**: Cplex Solution Pool for a Simple Facility Location Problem
  This example uses the incumbent checking call UserIncbCall as an advanced filter for accepting or rejecting solutions found by CPLEX.

### 4.43.6 Choosing an appropriate Solver

For any of the GAMS problem classes (LP, MCP, MINLP, ...), there is no solver that is best on every problem instance. Below, we provide some links to rules of thumb on choosing a solver or solver comparisons.


- M. Bussieck and S. Vigerske, **MINLP Solver Software**.


- H. Mittelmann: **Decision Tree for Optimization Software** and **Benchmarks for Optimization Software**

### 4.43.6.1 Relative Merits of MINOS and CONOPT

#### 4.43.6.1.1 How to choose between MINOS and CONOPT

It is almost impossible to predict how difficult it is to solve a particular model. The best and most reliable way to find out which solver to use is to try out both. However, there are a few rules of thumb:

**CONOPT** is well suited for models with very non-linear constraints. If you experience that MINOS has problems achieving feasibility during the optimization, you should try CONOPT. On the other hand, if your model has few nonlinearities outside the objective function, MINOS and QUADMINOS is probably the best solver.

**CONOPT** is has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom (this means: the number of variables is approximately the same as the number of constraints - in other words, models that are almost square). In these cases CONOPT is likely to outperform MINOS while for models with many more variables than equations MINOS is probably more suited.

**CONOPT** has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one, **CONOPT** will take advantage of this property. Similarly, intermediate variables only used to define objective function terms are eliminated from the model and the constraints are moved into the objective function.

**CONOPT** has many built-in tests (e.g. tests for detecting poor scaling). Many models that can be improved by the modeler are rejected with a constructive message. **CONOPT** is therefore a useful diagnostic tool during model development even if another solver is used for the production runs.
4.43.6.1.2 Why serious NLP modelers should have both MINOS and CONOPT

It is almost impossible to predict how difficult it is to solve a particular model. However, if you have two solvers, you can try both. The overall reliability is increased and the expected solution time will be reduced.

On a test set of 196 large and difficult models, many poorly scaled or without initial values, both MINOS and CONOPT failed on 14 models. However only 4 failed on both MINOS and CONOPT. So the reliability of the combined set of solvers is much better than any individual solver.

Many examples of poorly formulated models were observed on which MINOS failed. CONOPT rejected many of the models, but with diagnostic messages pinpointing the cause of the problem. After incorporating the changes suggested by CONOPT, both MINOS and CONOPT could solve the model. Switching between the two solvers during the initial model building and debugging phase can often provide useful information for improving the model formulation.

4.43.6.1.3 Special Offer for two NLP Solvers

In order to encourage modelers to have two NLP solvers, GAMS offers a 50% discount on the second solver when both MINOS and CONOPT are purchased together.

4.43.6.2 PATH versus MILES

This document describes some of the differences between the MCP solvers PATH and MILES. MILES is a free solver, that comes with the GAMS/BASE module, while PATH is an optional solver, that is charged for separately.

PATH and MILES are two GAMS solvers capable of solving mixed nonlinear complementarity problems (MCP). Both solvers are based on the sequential linear complementarity algorithm, i.e., they both solve a sequence of linear mixed complementarity problems whose solutions typically converge to the solution of the MCP. To solve each of the linear subproblems (major iterations), both codes use a generalization of an algorithm due to Lemke that is based on a sequence of pivots (minor iterations) similar to those generated by the simplex method for linear programming. To do these pivots efficiently, both codes use the same sparse linear algebra package.

As a result of the above similarities, the performance of the two codes is comparable for many "easy" models. Viewed over a broad range of problems, however, PATH is typically faster and more robust than MILES. While both codes solve all the MCP and MPSGE models in GAMSLIB, PATH significantly outperforms MILES on the MCPLIB test collection found at CPNET.

Most sophisticated MCP and MPSGE modelers prefer to use PATH over MILES. PATH has a crashing scheme that allows it to quickly improve the user given starting point before starting to solve the linear subproblems. This frequently speeds up solution time. PATH automatically attempts to fix "singular" models using a technique based on proximal perturbations. In many cases, this enables the linear subproblems to be solved, leading to a model solution. This typically helps modelers at model development time.

PATH has many more solution options to enable it solve difficult models. The code automatically tries useful options on difficult problems using a restart procedure. PATH has a much more sophisticated "globalization" procedure that typically improves speed and robustness. PATH implements a nonmonotone watchdog technique. Stalling is frequently circumvented by allowing larger steps to be taken toward solutions.

PATH has many more diagnostic features that help uncover problems in a model. In particular, singularities in the model, zero rows and columns and several measures of optimality are returned to the user. Theoretically, PATH has better convergence properties than MILES. In particular, new merit functions are known to allow more reliable and faster convergence.
4.44 The Save and Restart Feature

GAMS saves the information provided in input files in intermediate - mostly binary - files, called work files or scratch files. Some of these files are used to exchange information between GAMS and the solvers. They are usually deleted just before a GAMS run is complete. However, these intermediate files may be used to process an input file, save the result and later reload this file and continue with processing another input file. Thus, input files may be processed sequentially through the use of the intermediate files. This is a useful feature that can help to reduce the required time when, e.g., several runs of similar models are being made, all of them sharing an equal large initialization part.

The same process may be described in a different way. Assume a large GAMS program is run and an output file is generated, as usual. Suppose the large program is split in two parts. The first part is run and the resulting work file is saved along with the resulting listing file. Then the second part is run after reading in the data from the work file saved previously and a new listing file is generated for the second part. The content of the two listing files will be the same as the content of the output from the very first run when the large program was processed without interruption. Only the arrangement of the content will slightly differ. Splitting the files makes it possible to interrupt a GAMS task and restart it later without loss of information. Furthermore, changes could be made or errors corrected in the later parts.

Note
- The work file preserves all information (including declarations, values, option settings and dollar control options) known to GAMS at the end of the run.
- The work file is not machine specific and thus is portable between platforms. However, a work file that has been generated with one version of GAMS cannot be used for a restart with an older GAMS version (unless command line parameter forceWork is set).
- Absolute or relative path names (e.g. for put files) that do not exist or have the wrong directory separator may cause execution errors. Moreover, if the code in the work file executes a program that might not exist on another platform (e.g. gdxxrw on Linux) the GAMS execution might also result in execution errors.

This chapter illustrates the basics of the save and restart feature in Section Basic Usage and presents some use cases in Section Use Cases. Preventing unauthorized access to and obfuscating the content of work files is discussed in Sections Secure Work Files and Obfuscated Work Files, respectively. An overview of all command line parameters for saving and restarting in GAMS is given in Section Save and Restart Options (chapter The GAMS Call and Command Line Parameters).

4.44.1 Basic Usage

The mechanism to break up the compilation of a large input file into many components and stages is provided by the command line parameters save and restart.

The following demonstrates saves and restarts with the well known transportation model [TRNSPORT]. First, the code is split into three parts, resulting in the files tranmodel.gms, transolve.gms, and tranreport.gms. The file tranmodel.gms contains the first part of the model up to and including the model statement:

Sets
  i "canning plants" / seattle, san-diego /
  j "markets" / new-york, chicago, topeka / ;

Parameters
  a(i) "capacity of plant i in cases"
  / seattle 350
san-diego 600 /

b(j) "demand at market j in cases"
/ new-york 325
chipago 300
topeka 275 /;

Table d(i,j) "distance in 1000 miles"

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chipago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Scalar f "freight in dollars/case per 1000 miles" /90/;

Parameter c(i,j) "transport cost in $1000/case";
c(i,j) = f * d(i,j) / 1000;

Variables

x(i,j) "shipment quantities in cases"
z "total transportation costs in 1000$";
Positive Variable x;

Equations

cost "define objective function"
supply(i) "observe supply limit at plant i"
demand(j) "satisfy demand at market j";

cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;

The file transolve.gms contains the solve statement:
solve transport using lp minimizing z;

And the third file, tranreport.gms, contains the display statement:

display x.l, x.m ;

Observe that concatenating the three files (in the right order) results in the original model [TRNSPORT].

4.44.1.1 Saving The Work File

The information in tranmodel.gms may be stored by using the following call to GAMS:

> gams tranmodel save=s1

This command line statement creates the output file tranmodel.lst and the work file s1.g00 in the current working directory.

Note that the command line parameter s is a synonym to save.
4.44.1.2 Restarting from the Work File

Consider the following call:

> gams transolve restart=s1

GAMS reads the work file named s1.g00 and regenerates the information stored in tranmodel.gms. Then transolve.gms is run and the result is as if the a concatenation of tranmodel.gms and transolve.gms had been executed. Note, that the output file tranmodel.lst will contain the echo print of the model and transolve.lst will contain the echo print of the solve statement and all the output generated by the solve.

Note that the command line parameter r is a synonym of restart.

Observe that a restarted run always requires a GAMS input file to continue with. The restart does not alter work files. They may be used repeatedly to continue a particular run many times, possibly with many different continuation input files.

4.44.1.3 A Sequence of Saves and Restarts

In case there are more than two files that should be run sequentially, the second and any other non-final run will have to generate input files for a following restart. Therefore, their workfiles need to be saved.

Following the splitup of model [TRNSPORT] into the three files tranmodel.gms, transolve.gms, and tranreport.gms, a sequence of GAMS calls that would run the whole trnsport model could be as follows:

> gams tranmodel s=s1
> gams transolve r=s1 s=s2
> gams tranreport r=s2

The listing file of tranreport.gms will contain the compilation output with the echo print of the display statement, the final execution summary with the output generated from the display statement and the file summary. The listing files of all three input files together will have the same content as trnsport.lst (generated by a run of trnsport.gms).

Observe that the three files could also have been processed with a sequence of $include file statements. The advantages of using the save and restart feature instead are given in section Use Cases below.

4.44.1.4 Avoiding Common Mistakes

A common mistake that occurs when using the save and restart feature is running GAMS on the same file twice, e.g.,

> gams trnsport s=trans
> gams trnsport r=trans

In this case all the data and equation definitions are repeated, which will cause compilation errors for the second run as in GAMS each data item may be defined only once.

Further, it is the responsibility of the modeler to ensure that the contents of the input file matches that of the work file, although the compiler will issue errors if it detects any inconsistencies, such as references to symbols not previously declared.
4.44.1.5 Prefixing Symbols in the Work File

Assume writing some reporting code that works off a restart file and new symbol names that have not been used in the previous program are required. GAMS offers a convenient and systematic way to achieve this by prefixing all symbols in the work file with a specified string. Consider the following example, again using the transportation model \texttt{[TRNSPORT]}:

> gams trnsport s=prefixed symPrefix=aa_

This solves the model and saves the work file \texttt{prefixed.g00}. The command line parameter \texttt{symPrefix} has the effect that all symbols in the work file are prefixed with \texttt{aa}. For the next step, consider the following simple reporting code, saved in a file called \texttt{report.gms}:

Scalar i / 0 /;
loop(aa_i, i = i+1);
display 'number of canning plants', i;

Note that this code will be run with a restart from the work file \texttt{prefixed.g00}. Therefore, the looping set \texttt{aa_i} is identical to the set \texttt{i} in the model \texttt{[TRANSPORT]}. Since all symbols from the original model have been prefixed, convenient symbol names can be used for reporting purposes. The run of \texttt{report.gms} is achieved by the following call:

> gams report r=prefixed

The resulting listing file contains the following output:

---- 76 number of canning plants
PARAMETER i = 2.000

4.44.2 Use Cases

The basic function of a work file is to preserve information that has been expensive to produce. The following discusses several use cases for work files.

4.44.2.1 Separation of Model and Data

The separation of model and data is one of the core principles of the GAMS modeling paradigm. Using the save and restart feature helps to exploit this separation.

Separation of model and data will be illustrated on the transportation model \texttt{[TRNSPORT]}. First, consider a GAMS file \texttt{transportmodel.gms} which contains only the algebraic representation of the transportation problem, obtained by removing all data and execution statements from \texttt{[TRNSPORT]}. 

---
Sets i canning plants
j markets

Parameters a(i) "capacity of plant i in cases"
b(j) "demand at market j in cases"
c(i,j) "transport cost in 1000$/case"
d(i,j) "distance in 1000 miles"
Scalar f "freight in $/case per 1000 miles"

Variables x(i,j) "shipment quantities in cases"
z "total transportation costs in 1000$"
Positive Variable x

Equations cost "define objective function"
supply(i) "observe supply limit at plant i"
demand(j) "satisfy demand at market j"

\[
\text{cost ..} \quad z = \text{e} = \sum_{i,j} c(i,j) \times x(i,j)
\]
\[
\text{supply(i) ..} \quad \sum_j x(i,j) = \text{l} = a(i)
\]
\[
\text{demand(j) ..} \quad \sum_i x(i,j) = \text{g} = b(j)
\]

Model transport /all/ ;

Second, consider a GAMS file transportdata.gms that contains the data of the model as well as the solve and display statements:

Sets i / seattle, san-diego /
j / new-york, chicago, topeka / ;

Parameters a(i) / seattle 350
san-diego 600 /
b(j) / new-york 325
chicago 300
topeka 275 /
Table d(i,j)
         new-york  chicago  topeka
seattle    2.5     1.7     1.8
san-diego  2.5     1.8     1.4

Scalar f / 90 / ;
c(i,j) = f * d(i,j) / 1000 ;
Solve transport using lp minimizing z ;
Display x.l, x.m ;

The second file (transportdata.gms) cannot be run alone as the definition of model transport is missing. However, one may first run the first file (transportmodel.gms) and save its work file. Then one can restart from this work file to run the second file:

> gams transportmodel.gms s=transmod
> gams transportdata.gms r=transmod
4.44.2.2 Advanced Separation of Model and Data

In the previous example some execution time statements namely the assignment of c, the solve, and the display where performed in the data file. If the model execution logic is more complex we do not want to add this to the data file. Hence we create in this example a restart file from this first file but will only compile, but not execute (see difference between compile and execute phases in section GAMS Compile Time and Execution Time Phase):

```gams
$onEmpty
Sets i(*) canning plants / /  
j(*) markets / /  
Parameters a(i) "capacity of plant i in cases" / /  
b(j) "demand at market j in cases" / /  
c(i,j) "transport cost in 1000$/case" / /  
d(i,j) "distance in 1000 miles" / /  
Scalar f "freight in $/case per 1000 miles" / 0 /;  
c(i,j) = f * d(i,j) / 1000 ;  
Variables x(i,j) "shipment quantities in cases"  
z "total transportation costs in 1000$" ;  
Positive Variable x ;  
Equations cost "define objective function"  
supply(i) "observe supply limit at plant i"  
demand(j) "satisfy demand at market j" ;  
cost .. z =e= sum((i,j), c(i,j)*x(i,j)) ;  
supply(i) .. sum(j, x(i,j)) =l= a(i) ;  
demand(j) .. sum(i, x(i,j)) =g= b(j) ;  
Model transport /all/ ;  
Solve transport using lp minimizing z ;  
Display x.l, x.m ;  

Second, we consider a file that contains the data of the model but no other execution time statements. Because we already have empty data statements (/ /) for the data items in the first file, we need to instruct the compiler to allow a second data statement with the real data using $onMulti:

```gams
$onMulti
Sets i / seattle, san-diego /  
j / new-york, chicago, topeka / ;  
Parameters a(i) / seattle 350  
           san-diego 600 /  
b(j) / new-york 325  
           chicago 300  
           topeka 275 /  
Table d(i,j)  
    new-york  | chicago  | topeka  
    seattle   | 2.5      | 1.7     | 1.8  
    san-diego | 2.5      | 1.8     | 1.4  ;
```
Scalar f / 90 / ;
$offMulti

In order to run the model we first compile, but not execute the first model (see command line parameter action=c) and create a save file. Next, we run the second model. This does a continued compilation, it compiles the data statements from the second files and then executes the execution time statements from the first (c(i,j)=..., solve ..., and display ...) and second (here there are none).

> gams transportmodel.gms action=c s=transmod
> gams transportdata.gms r=transmod

4.44.2.3 Generating Concise Listing Files

By default a GAMS listing file has many components, see chapter GAMS Output for details. In case a more concise listing file is needed, the save and restart feature may be used to generate such a file.

Recall the reorganization of the transportation model [TRNSPORT] into a model file trnsportmodel.gms and a data-and-solve file trnsportdata.gms from the previous section. Now consider the addition of two further files. The first one, called trnsportreport.gms, contains post-solution computations for reporting:

Parameter m(*,*), 'movement of commodities in cases';
  m(i,j) = x.l(i,j);
  m('total',j) = sum(i, x.l(i,j));
  m(i,'total') = sum(j, x.l(i,j));
  m('total','total') = sum(j, m('total',j));

The second one, called trnsportdisplay.gms, contains only a display statement:

Option decimals = 0;
Display m;

Using these four files, the following save and restart sequence can be called:

> gams trnsportmodel s=trans1
> gams trnsportdata r=trans1 s=trans2
> gams trnsportreport r=trans2 s=trans3
> gams trnsportdisplay r=trans3

The output file trnsportdisplay.lst is brief. Apart from the echo print, the execution time and the file summary, it contains only the output generated by the display statement:

---- 52 PARAMETER m movement of commodities in cases

new-york    chicago   topeka    total
seattle      50        300       275      350
san-diego    275       300       275      550
total        325       300       275      900

In this way it is possible to create output files that are concise and contain only the information needed, while at the same time the more detailed output is stored in other listing files and may be inspected if needed.
4.44.2.4 Incremental Program Development

GAMS programs are often developed in stages. A typical approach is to start with set statements, tables and data manipulations, then equations are declared and defined, followed by model and solve statements and finally assignments for generating reports. As each piece of the model is built, it should be run and checked for errors by inserting diagnostic display and abort statements. As confidence grows that the parts of the model are correct, it is useful to save the completed parts in a work file. Subsequently, it is possible to work only on the piece under active development by restarting from the saved work file and thus reducing running time and the amount of output produced in each of the development runs.

This approach is especially useful when working on the results report part, since the setup and solution of a model instance is typically dominating the computing time, while the report part has to be run often to get all details of setting up content and layout into a satisfying form. Thus, the model may be generated and solved and the result saved in a work file. One may then restart from the work file while developing the report.

4.44.2.5 Tracking a Sequence of Difficult Solve Statements

In many cases where solves are known to be difficult and expensive, it may be too risky to let GAMS process a job containing many solve statements. The risk is that if one solve does not proceed to normal completion, then the following solve will not be possible or will start from a bad initial point and much time and effort will be wasted.

An alternative is to request one solve at a time and save the work file. By doing so, the output of each solve can be carefully inspected before proceeding. If everything is as expected, the job can be restarted and the next solve be executed. If an error has occurred, the previous solve may be repeated, maybe with a different initial point or modified working limits such as iteration or time limits.

4.44.2.6 What-If Analysis

Many modeling exercises involve a 'what if' analysis, in which a base case is defined and the point of the study is to see how the system changes when circumstances change, either naturally or by design. Often, the effect of many different changes to the base case are considered separately.

The save and restart feature facilitates such analyses. The base case may be saved using a work file and then all desired scenarios may be run separately by restarting from the same work file. Each scenario probably involves only doing some changes to the data (e.g., coefficients in equations or variable bounds), solving the changed model (the solution of the base case will then automatically be used as a starting point), and reporting.

4.44.2.7 The GAMS Runtime License

A GAMS runtime license can be used to restrict users to run an existing model with their own data without the possibility to modify the model itself.

For example, recall the separation of model [TRNSPORT] from Section Separation of Model and Data above into a model file trnsportmodel.gms and a data and solve file trnsportdata.gms. The developer of the model may run trnsportmodel.gms with the command

```gams
> gams trnsportmodel s=trans
```

to generate a work file trans.g00. She then distributes this work file and the data and solve file trnsportdata.gms to endusers. If the endusers have a run-time license for GAMS, they are not able to see the model or change it by adding any new variables or equations, but they can run the model by the command

```gams
> gams trnsportdata r=trans
```

Observe that the endusers have full control of the data, e.g., are able to manipulate the elements of sets or the values of the various parameters.
4.44.3 Secure Work Files

When models are distributed to users other than the original developers or embedded in applications to be deployed by other developers, issues of privacy, security, data integrity and ownership arise. One may have to hide, protect or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature. Examples include the following:

Privacy

A Social Accounting Matrix supplied by a statistical office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

Security

Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. However, the final model will be used at different locations around the world.

Integrity

Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information need to be protected from accidental changes that would compromise the operation of the model.

To address these issues, so called secure work files have been introduced to GAMS. Such a work file behaves like any other work file but it is locked to a specific user’s license file. The content of a secure work file protected against unauthorized access via the GAMS license mechanism.

Attention

A special GAMS license is required to create secure work file, see also Usage. The security features of secure work files are not extended to the solver level. As a consequence, full information about a model instance can be extracted on the GAMS solver level by a user that is authorized to solve the model. See Limitations for more information.

4.44.3.1 An Introductory Example

The well-known transportation model [TRNSPORT] will be used again to illustrate the creation and deployment of a secure work file. Assume one wants to distribute this model but there are concerns about proprietary formulations and data. In addition, one would like to prevent that the user does unintentional modifications to the model. It is assumed that the objective function and the supply constraints are to be hidden from other users and only the demand figures should be allowed to be changed. Data that is not needed any more will be purged as well.

First, a copy of the model from the model library is created, the model is run and a normal work file t1 is created:

> gamslib trnsport
> gams trnsport s=t1
Next, a file t2.gms with access control commands is created,

```
$eolcom //
$protect all // make all symbols read only
$purge  d f // remove items d and f
$hide  cost supply a // make objective invisible
$expose transport b // allow changes to b
```

and a secure work file t2.g00 is created by executing t2.gms with a restart from t1.g00:

```
> gams t2 r=t1 s=t2 plicense=target
```

The newly created work file is secure since the access control commands were activated with the privacy GAMS license option PLicense. This command line parameter specifies the name of the target user's license file. The effect is that the work file t2.g00 can now only be read with the target license file.

The log output will contain the following lines:

```
GAMS Rev 124  Copyright (C) 1987-2001 GAMS Development...
Licensee: Source User Name
         Source Company Name
*** Creating a Secure Restart File for:
***     Target User Name
***     Target Company Name
--- Starting continued compilation
--- T2.GMS(6)  1 Mb
--- Starting execution
*** Status: Normal completion
```

The three lines starting with *** are a recap of the content of the target license file. From now on, the source and the target licenses are 'burned into' this file and all its descendants. One can now send the restart file to the target user or system.

The target user may run the model with new data, add new GAMS statements and create new work files. However, there are two restrictions: some of the symbols are hidden and this model can only be executed using the target license file.

For example, the target user may want to half the demand and compare the original solution with the new one. The following file, called t3.gms, will accomplish this:

```
Parameter rep 'summary report';
rep(i,j,'base') = x.l(i,j);
b(j) = b(j)*0.5;
solve transport minimizing z using lp;
rep(i,j,'half') = x.l(i,j);
display rep;
```

This new file may be executed on the target system, restarting from the work file t2.g00:

```
> gams t3 r=t2
```
The resulting log file will contain the following lines:

GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Target User Name
  Target User Company
*** Restarting from a Secure Restart File created by:
***   Source User Name
***   Source Company Name
--- Starting continued compilation
--- T3.GMS(5) 1 Mb
...

Note that the originator/owner of the secure work file is mentioned by name. A similar message will be contained in the listing file:

EXECUTION TIME = 0.000 SECONDS 1.1 Mb WIN201-124

**** Secure Save/Restart File Source:
   Source User Name
   Source Company Name
**** Secure Save/Restart File Target:
   Target User Name
   Target User Company
...

A more detailed inspection of the listing file shows that the hidden variables and equations do not appear in the usual equation and variable listings and the solution print. The hidden items can only be accessed via a public (exposed) model and a solve statement. However, note that the full model instance may still be accessed by the target user, see Limitations.

In the following, secure work files and the access control commands are described in more detail.

4.44.3.2 Usage

Secure work files control access to symbolic and numeric information and can only be read by a specific GAMS user. The initial creation or additions to access control requires a special GAMS license. Saving secure work files without new access controls does not require a special GAMS license. The creation or addition of access control is signaled by the use of the GAMS command line parameter PLicense, which gives the name of a privacy license file. The shortcut PLICENSE=LICENSE sets the privacy license to the current license file. This is convenient when experimenting with access controls.

When a secure work file is written for the first time, the first and second lines of the current license file and the privacy license file are inserted into the work file. This information cannot be changed any more and the original source and the intended target users are locked into the work file.

A secure work file may be used just like any other work file and new work files may be derived from secure files. However, their use is restricted to the target user specified with the command line parameter PLicense. The target user can, if licensed, add access controls to an existing secure file by using the parameter PLICENSE=LICENSE but cannot change the original information about source and target users.

There are four access control commands (ACCs) that are processed during the compilation phase. These commands can be inserted anywhere in the program. They are processed in chronological order and have the following syntax:

$acc ident1 ident2 ...
$acc all

Here ident1 and ident2 are GAMS identifiers previously defined in the program and acc denotes one of the four access control commands:
<table>
<thead>
<tr>
<th>Dollar Control Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge</td>
<td>Removes the objects and all data associated.</td>
</tr>
<tr>
<td>hide</td>
<td>Hides the objects but allows them to be used in model calculations.</td>
</tr>
<tr>
<td>protect</td>
<td>The objects cannot be modified but they may be used in model calculations.</td>
</tr>
<tr>
<td>expose</td>
<td>Removes all privacy restrictions, the symbols will be reverted to their original state.</td>
</tr>
</tbody>
</table>

The keyword `all` applies the ACCs to all identifiers defined up to this point in the GAMS source code. Note that ACCs may be changed and redefined within the same GAMS program. However, identifiers inherited from a restart file cannot be changed.

### 4.44.3.3 A Practical Example

This section uses the transportation model `[TRNSPORT]` to show how to hide input data and results from the target user. The target user will be allowed to view percentage changes from an unknown base case only. In addition to the original model, a data initialization and a report model will be introduced.

First, a method to calculate input data is defined. As the GAMS language does not offer the definition of methods (or functions), here a model is used to define algebraically the desired correspondence between the input and output of the method we wish to emulate. Execution of the method will then correspond to solving the model. The model is the following:

```plaintext
$include trnsport.gms

Variable newc(i,j) 'new transport data';
Equation defnewc(i,j) 'definition of new transport data';
defnewc(i,j).. newc(i,j) =e= f*d(i,j)/1000;
Model getc 'compute new transport data' / defnewc /;
Solve getc using cns;
```

By solving model `getc` (see Constrained Nonlinear System (CNS) for details on problem type CNS), the variable `newc(i,j)` will obtain the value of `f*d(i,j)/1000` in the variable level attributes. Thus, `newc.l` corresponds to parameter `c` in the original model.

Next, the objective function of the original model is changed to reflect **economies of scale**. Furthermore, a base case value `basex` is computed for later use in the reporting model:

```plaintext
Scalar beta 'scaling exponent' / 1.1 /;
Equation newcost 'economies of scale objective function';
newcost.. z =e= sum((i,j), newc.l(i,j) * x(i,j)**beta);
Model estrans / newcost, supply, demand /;
solve estrans using nlp minimizing z;
```

Parameter `basex(i,j)` 'base values of x';
`basex(i,j)` = `x.l(i,j)`;

Finally, a method to transform the results of model `estrans` to the relative change with respect to the base case is defined. As for the computation of the input data (`newc`), a model is used to emulate this method:
Variable delta(i,j)  'percentage change from base values';
Equation defdelta(i,j)  'definition of delta';
defdelta(i,j)$basex(i,j).. delta(i,j) =e= 100*(x.l(i,j)-basex(i,j))/basex(i,j);

Model rep / defdelta /;
solve rep using cns;

Assume the GAMS code above has been saved in a file p1.gms. Running GAMS on this file,

> gams p1 s=p1

creates a work file with the name p1.g00.

In the following, some test runs similar to those that are expected to be defined by the target user are made. Three scenarios to be solved in a loop are defined in file u1.gms:

Set   s / one, two, three /;
Parameter sbeta(s) / one 1.25, two 1.5, three 2.0 /
    sf(s) / one 85, two 75, three 50 /;
Parameter report  'summary report';
loop(s,
    beta = sbeta(s);
    f   = sf(s);
    solve getc using cns;
    solve estrans using nlp minimizing z;
    solve rep using cns;
    report(i,j,s) = delta.l(i,j);
    report('','beta',s) = beta;
    report('','f',s) = f;
    report('obj','z',s) = z.l
);

display report;

File u1.gms can be executed with a restart from the work file p1.g00:

> gams u1 r=p1

The display statement generates the following output:

---- 109 PARAMETER report summary report

<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
<th>three</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle.new-york</td>
<td>-4.050</td>
<td>-6.967</td>
<td>-8.083</td>
</tr>
<tr>
<td>seattle.topeka</td>
<td>233.958</td>
<td>348.468</td>
<td>404.187</td>
</tr>
<tr>
<td>san-diego.new-york</td>
<td>3.605</td>
<td>6.201</td>
<td>7.194</td>
</tr>
<tr>
<td>san-diego.chicago</td>
<td>28.138</td>
<td>40.719</td>
<td>47.228</td>
</tr>
<tr>
<td>.beta</td>
<td>1.250</td>
<td>1.500</td>
<td>2.000</td>
</tr>
<tr>
<td>.f</td>
<td>85.000</td>
<td>75.000</td>
<td>50.000</td>
</tr>
<tr>
<td>obj .z</td>
<td>526.912</td>
<td>1652.963</td>
<td>13988.774</td>
</tr>
</tbody>
</table>
Note that all symbols are still completely exposed. Access controls need to be added to the model p1.gms before it can be made available to the target client. The information to be protected is the original distance matrix and derived information. A recommended procedure is to first hide everything and then enable access to only selected parts of the model. The access control information is collected in the file s1.gms:

```
$hide all
$expose getc estrans rep
$expose i j z delta
$expose f beta a b
```

Using the initial workfile p1, a secure work file s1.g00 is created by executing

```
> gams s1 r=p1 s=s1 plicense=license
```

To test the system from the target user's point of view, the license of the current GAMS system is used also for the target user.

To test the secure work file, problem u1.gms is run again, restarting from the work file s1:

```
> gams u1 r=s1
```

Inspecting the resulting listing file, one observes that equation, variable and solution listings related to the hidden variables are not shown anymore. Any attempt to reference a hidden variable will cause a compilation error.

### 4.44.3.4 Limitations

One of the design goals for secure work files has been to minimize the impact on other components of the GAMS system. Solvers used within a secure environment should work as if called within a normal environment. This implies that certain information about a model can easily be recovered by using solvers like CONVERT or solver options like writelp for CPLEX, writemps for CBC, or gams/interactive for SCIP.

For example, consider the secure work file t2.g00 from Section An Introductory Example. The target user could now reproduce the complete model instance, including the hidden equation supply or the objective function cost by running the following GAMS code, restarting from t2.g00:

```gams
option mip = SCIP;
transport.optfile = 1;
$echo gams/interactive = "display prob quit" > scip.opt
solve transport min z using mip;
```

This produces the log output:
original problem has 6 variables (0 bin, 0 int, 6 cont) and 5 constraints

STATISTICS
  Problem name : r
  Variables : 6 (0 binary, 0 integer, 0 implicit integer, 6 continuous)
  Constraints : 0 initial, 5 maximal

OBJECTIVE
  Sense : minimize

VARIABLES
  [continuous] <x(seattle,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(seattle,chicago)>: obj=0.153, original bounds=[0,+inf]
  [continuous] <x(seattle,topeka)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(san-diego,chicago)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,topeka)>: obj=0.126, original bounds=[0,+inf]

CONSTRAINTS
  [linear] <supply(seattle)>: <x(seattle,new-york)>[C] +<x(seattle,chicago)>[C] +<x(seattle,topeka)>[C] <= 350;
  [linear] <supply(san-diego)>: <x(san-diego,new-york)>[C] +<x(san-diego,chicago)>[C] +<x(san-diego,topeka)>[C] <= 600;
  [linear] <demand(new-york)>: <x(seattle,new-york)>[C] +<x(san-diego,new-york)>[C] >= 325;
  [linear] <demand(chicago)>: <x(seattle,chicago)>[C] +<x(san-diego,chicago)>[C] >= 300;
  [linear] <demand(topeka)>: <x(seattle,topeka)>[C] +<x(san-diego,topeka)>[C] >= 275;

END

...
Note, that the command line parameter action=compile is used to only compile, but not execute, the model. Observe that the name of the regular work file is *named.g00 and the name of obfuscated work file is *obfuscated.g00.

In the next step, the obfuscated work file is moved to a non-secure machine and GAMS is restarted from this work file on an empty GAMS model file. This executes the original model (here *trnsport), but with all names obfuscated. The outcome of this execution is saved in another, still obfuscated, work file *obfuscated.g00.

The work file *obfuscated.g00 is now brought back to a trusted machine. To obtain the results, say the marginal values of the supply equation, in a non-obfuscated form, GAMS is executed again on a model that writes out values of supply.m to a GDX file supply.gdx. This model is restarted from the obfuscated work file *obfuscated.g00, but additionally the originally created non-obfuscated work file *named.g00 is passed in via the command line parameter RestartNamed (short: *rn):

Recall, that the work file *named.g00 resulted from the initial compilation and contains the names from the original namespace. It never left the trusted environment. The effect of combining options *r and *rn is that all content from the obfuscated work file is taken, except for the names of symbols and labels, the explanatory texts, and the listing file titles and subtitles, which are read from the work file specified via *rn.

GAMS automatically performs the following three checks to ensure that the named and obfuscated work files are consistent:

1. The number of labels and symbols must be identical.
2. The size of the string pool must be identical.
3. The first 10 labels point to the same addresses in the string pool.

The first two checks imply that the execution in the obfuscated name space cannot introduce new symbols or labels or even new strings (e.g., from display *this is a new string*). Therefore, the empty GAMS program that was used to execute from the obfuscated work file above can hardly be replaced by any useful code since the obfuscated symbols and labels are not known.
4.45 Embedded Code Facility

4.45.1 Motivation

GAMS uses relational data tables as a basic data structure. With these, GAMS code for parallel assignment and equation definition is compact, elegant, and efficient. However, traditional data structures (arrays, lists, dictionaries, trees, graphs, ...) are not natively or easily available in GAMS. Though it is possible to represent such data structures in GAMS, the GAMS code working with such structures can easily become unwieldy, obfuscating, or inefficient. Also, in the past it was not easy to connect libraries from other systems for special algorithms (e.g. graph algorithms, matrix operations, ...) to GAMS without some data programming knowledge and a deep understanding of internal representation of GAMS data (e.g. the GDX API).

The Embedded Code Facility addresses this need and extends the connectivity of GAMS to other programming languages. It allows the use of external code (e.g. Python) during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary. It utilizes an API (this API can be found in apifiles/C/api/emlib*.h together with the code for the Python and Connect embedded code library apifiles/C/api/embpyoo.c/h) and additional source code for common tasks so that the user can concentrate on the task at hand and not the mechanics of moving data in and out of GAMS.

4.45.2 Concept

As pointed out in section Motivation, the main idea of the Embedded Code Facility is to enable the use of external code in GAMS and give this code direct in-memory access to the GAMS database (or better: to GAMS symbols, namely sets, parameters, variables, and equations). This can be done by defining sections in the GAMS code which contain no GAMS code, but code written in a defined external code. These sections can be used at both GAMS compile and execution time (compare section GAMS Compile Time and Execution Time Phase). Details about how to do this can be found in the Syntax section.

Note

We will continuously extend this feature to different programming languages. At the moment only Python and Connect are supported.

Also, the system provides some help to develop and debug the external code independent of GAMS first. More about this topic can be found in section Troubleshooting Embedded Python Code.

The communication of the data between the GAMS part of the program and the embedded code part was inspired by the existing interface to GDX in many ways. For example, using dollar control options $on/offMulti, $onMultiR, and $on/offFiltered it is possible to decide if data changed in the embedded code should replace the GAMS data or get merged with it and just like loading data from GDX, one can decide if data from embedded code should change the GAMS database filtered or domain checked.
4.45.3 Simple Example

4.45.3.1 Python

In a very first example we look into some Python code that helps to split some label names that are already present in GAMS. We do this at compile time in order to read the broken up pieces as individual sets (country and city) into GAMS plus some mapping sets (mccCountry and mccCity) between the original labels and the new labels. The compile-time embedded code starts with a $onEmbeddedCode followed by the type of code to expect (Python). The lines between $onEmbeddedCode and $offEmbeddedCode is Python code. We do not want to go into Python details, but the first few lines initialize some empty Python list objects (mccCountry and mccCity) as well as some empty Python set objects. In the Python for loop that follows we iterate over all individual labels of the GAMS set cc. Python gets access to the GAMS set cc via the member function get of the implicitly defined Python object gams. get returns an object that is iterable and can be used in a Python for loop. The type of the records one gets depend on the dimensionality and type of the GAMS symbol. In the loop body we use the Python split function to extract the first (r[0] is country) and second (r[1] is city) part of the label. The three strings cc, r[0], r[1] are used to build up the Python list objects that store the information for the maps mccCountry and mccCity and the Python set objects that store the labels for the new sets country and city. The Python set has the advantage to store a label just once even if we add it multiple times. Python prepares to send items back to GAMS via the gams member function set that can deal with both Python list and Python set objects. The command $offEmbeddedCode is followed by a list (without separating commas) of GAMS symbols that instructs the GAMS compiler to read these symbols back. This list is optional and can be left out. GAMS will import all symbols set by the Python code. The GAMS compiler (without executing execution-time embedded code) does not know which symbols will be implicitly loaded, hence it cannot define symbols. If such a symbol is referenced further down in the compilation process, the compiler will complain with error $141: Symbol declared but no values have been assigned. One can avoid this situation by explicitly list the symbols on the end/pauseEmbeddedCode line.

Another caveat with implicit loading is related to aliases. GAMS does not allow to load data into an alias, so the following code will fail with a compilation error 493 Alias cannot receive data:

Set i(*); Alias (i,j);
$onEmbeddedCode Python:
gams.set('j',['i1','i2','i3'])
$offEmbeddedCode j

By leaving off the symbol j from the $offEmbeddedCode line, the code works fine and actually imports the set i. The gams.set and other write methods in embedded code to an alias symbol, here j, actually fills the aliased set, here i, and hence the implicit load will load the modified symbol i into GAMS.

Implicit loading also needs special consideration in the context of external input. Since embedded code inside an $on/offExternalInput section is not executed if the command line parameter IDCGDXInput has been set, there will be no symbols to be loaded implicitly. So make sure the symbols to load are given explicitly in such a situation.

Note that GAMS syntax is case insensitive while Python is case sensitive. Hence, the strings that represent the GAMS symbol names in the Python code can have any casing (e.g. gams.get("cc") or gams.get("CC")) , while the corresponding Python objects need to have consistent casing throughout the Python code.

The following code presents the entire embeddedSplit example from the GAMS Data Utilities Library:
Set cc / "France - Paris", "France - Lille", "France - Toulouse"
    , "USA - Washington DC", "USA - Houston", "USA - New York",
    , "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
    country
city
mccCountry(cc,country<) Mapping between country and related elements in set cc
mccCity(cc,city<) Mapping between city and related elements in set cc;
$m onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
r = str.split(cc, " - ", 1)
mccCountry.append((cc,r[0]))
mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;

The display in the listing file looks as follows:

---- 25 SET country
Spain , USA , Germany, France

---- 25 SET city
Berlin , Bilbao , Cordoba , Madrid
New York , Washington DC, Paris , Houston
Munich , Lille , Seville , Bonn
Toulouse

---- 25 SET mccCountry
France - Paris .France
France - Lille .France
France - Toulouse .France
Spain - Madrid .Spain
Spain - Cordoba .Spain
Spain - Seville .Spain
Spain - Bilbao .Spain
USA - Washington DC.USA
USA - Houston .USA
USA - New York .USA
Germany - Berlin .Germany
Germany - Munich .Germany
Germany - Bonn .Germany

---- 25 SET mccCity
France - Paris .Paris
France - Lille .Lille
France - Toulouse .Toulouse
Spain - Madrid .Madrid
Spain - Cordoba .Cordoba
Spain - Seville .Seville
Spain - Bilbao .Bilbao
The second example demonstrates the use of embedded code at execution time. The syntax for the execution time embedded code in this example is identical to the compile time variant with the exception of the keywords that start and end the embedded code section: `embeddedCode` and `endEmbeddedCode`. It is important to understand that the execution of the code happens at GAMS execution time, so e.g. no new labels can be produced and send back to GAMS. In this example we use some Python code to generate a random permutation of set elements of set `i` and store this in a two dimensional set `p`. In this example we do not use a loop to iterate through the elements of a GAMS system but make use of the fact that the Python object returned by `gams.get("i")` is iterable and can in its entirety be stored in the Python list with name `i` with the short and powerful command `i = list(gams.get("i"))`. The permutation of elements in list `p` which is a copy of list `i` is created by the Python statement `random.shuffle(p)`. The following code presents the entire example:

```python
Set i /i1*i10/
  p(i,i) "permutation";

embeddedCode Python:
import random
i = list(gams.get("i"))
p = list(i)
random.shuffle(p)
for idx in range(len(i)):
  p[idx] = (i[idx], p[idx])
gams.set("p", p)
endEmbeddedCode p

option p:0:0:1;
display p;
```

The display in the listing file looks as follows:

```
---- 11 SET p permutation
  i1 .i1
  i2 .i7
  i3 .i5
  i4 .i2
  i5 .i10
  i6 .i6
  i7 .i4
  i8 .i18
  i9 .i18
  i10.i13
```

### 4.45.3.2 Connect

In the following example, we need to process a CSV file that cannot be read directly by GAMS under `$onDelim`. So we instruct `Connect` to read the CSV file into the Connect database and then write to GAMS:
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onText
Read a CSV file that looks as follows:

date;symbol;price
2016/01/04;AAPL;105.35
2016/01/04;AXP;67.59
2016/01/04;BA;140.50
...

and can’t be read directly by GAMS under $onDelim because some labels include ‘/’ and are not quoted, use ‘;’ as field separator, and ‘,’ as decimal character.
$offText

$onEmbeddedCode
Connect:
- CSVReader:
  file: stockprice.csv
  name: stockprice
  indexColumns: [1, 2]
  valueColumns: [3]
  fieldSeparator: ‘;’
  decimalSeparator: ‘,’
- GAMSWriter:
  symbols:
  - name: stockprice
$offEmbeddedCode

display stockprice;

We leave the CSV file processing to the CSVReader and then export the stockprice symbol to GAMS. Please note that the symbols written to GAMS by the GAMSWriter do not have to be explicitly listed on the $offEmbeddedCode line, this is optional. Moreover, the Connect agents GAMSReader and GAMSWriter are only available under embedded code.

We might use the data to calculate an optimal portfolio. The share of each stock in this optimal portfolio is represented by the parameter share(stocks) and can be exported to Excel using the PandasExcelWriter. Before exporting to Excel, the data first needs to be transferred from GAMS to Connect using the GAMSReader:

* Continues the code from the previous example

Parameter share(stocks);
* For demonstration we fill share with random numbers
share(stocks) = uniform(0,1);

embeddedCode
Connect:
- GAMSReader:
  symbols:
  - name: share
- PandasExcelWriter:
  file: share.xlsx
  symbols:
  - name: share
    range: Share!A1
endEmbeddedCode
4.45.4 Syntax

This section explains the GAMS functions/keywords which were introduced to enable the Embedded Code Facility. The first subsection deals with the syntax for compile time, the second with the syntax for execution time (compare section GAMS Compile Time and Execution Time Phase).

4.45.4.1 Compile Time

There are three dollar control options to start an embedded code section at compile time:

\[
\begin{align*}
\$onEmbeddedCode[.tag] & \text{[Connect|Python]: [arguments]} \\
\$onEmbeddedCodeS[.tag] & \text{[Connect|Python]: [arguments]} \\
\$onEmbeddedCodeV[.tag] & \text{[Connect|Python]: [arguments]}
\end{align*}
\]

Lines following one of the above statements are passed on to the embedded code engine (e.g. Connect or the Python interpreter) until this dollar control option, which ends the embedded code section at compile time, is hit:

\[
\$offEmbeddedCode[.tag] \{symbol[<=]embSymbol[.dimX]\}
\]

These dollar control options are explained here in more detail. The usage of the optional arguments by the embedded code is explained in the specific embedded code engine section below.

Note

- The optional arguments from the \$onEmbeddedCode[S|V][.tag] statement can be accessed as gams.arguments in the Python code.
- The optional output symbols from the \$offEmbeddedCode[.tag] statement need to be written to by the embedded code.
- More about the specific GAMS Python syntax can be found below.

4.45.4.2 Execution Time

At execution time an embedded code section is started with one of these statements:

\[
\begin{align*}
\text{embeddedCode[.tag]} & \text{[Connect|Python]: [arguments]} \\
\text{embeddedCodeS[.tag]} & \text{[Connect|Python]: [arguments]} \\
\text{embeddedCodeV[.tag]} & \text{[Connect|Python]: [arguments]}
\end{align*}
\]

Similar to the compile time alternatives \$onEmbeddedCode[S|V][.tag], the first two variants are synonyms which allow parameter substitution in the code that follows, while the last variant does not allow this but passes the code verbatim to the embedded code engine. The usage of the optional arguments by the embedded code is explained in the specific embedded code engine section below. The arguments passed to the embedded code engine at execution time can be extended by the put_utility command ECArguments. For example,

\[
\begin{align*}
\text{put\_utility 'ECArguments' / 'world...'}; \\
\text{embeddedCode} & \text{ Python: Hello} \\
\text{gams.printLog(gams.arguments)} \\
\text{endEmbeddedCode}
\end{align*}
\]
prints Hello world....

Lines following one of the statements `embeddedCode[S|V]` are passed on to the embedded code engine (e.g. the Python interpreter) until one of the following two statements, which end the embedded code section at execution time, is hit:

```
endEmbeddedCode[.tag] {output symbols}
pauseEmbeddedCode[.tag] {output symbols}
```

When the optional `tag` suffix (which can be any string) is used to start an embedded code section, the same one has to be used to end/pause that section. Both statements end the embedded code section and switch back to GAMS syntax in the following lines. Also, both statements can be followed by a GAMS symbol or a list of GAMS symbols which would get updated in the GAMS database after the embedded code got executed. If `output symbols` are specified, they need to be written to (e.g. by `gams.set` for embedded Python code) by the embedded code before.

To continue a previously paused embedded code section one of the following statements is used:

```
continueEmbeddedCode [handle]: [arguments]
continueEmbeddedCodeS [handle]: [arguments]
continueEmbeddedCodeV [handle]: [arguments]
```

As seen before, the first two variants are synonyms which allow parameter substitution in the embedded code that follows, while the last variant does not allow this but passes the code verbatim to the interpreter. Again, the usage of the optional `arguments` by the embedded code is explained in the specific embedded code engine section below.

New in these statements is the optional `handle`. If omitted, the last code section that was paused will be continued. However, sometimes one might need to maintain different embedded code sections active in parallel and independent of each other. There is a new function to store a handle of the last embedded code section that was executed which could then later be used to continue a specific paused code section:

```handle = embeddedHandle;
```

What it exactly means to end or pause/continue an embedded code section depends on the embedded code engine and is explained in more detail with the code engine below. Independent of the embedded code engine it is important to note that if GAMS stops and restarts while executing a `solve` statement (with `solveLink=0`) continuing any embedded code section causes the code to fail with the following error message:

```
Error executing "continueEmbeddedCode" section
--- (Hint: "Solve" with SolveLink=0 frees previously initialized embedded libraries):
Error at line 71: No embedded library initialized
```

This happens because under `solveLink=0` the GAMS process terminates when the solver starts up. This causes the loss of the state of the embedded code environment that is linked into the GAMS process as a shared library. Under all other settings of `solveLink` the GAMS process is not stopped and restarted and hence a paused embedded code environment can be continued after the solve statement has carried out.
4.45.5 Python

GAMS and Python complement each other in many different ways. GAMS has a compact but readable syntax for data assignment and equation definition statements. Python is a great scripting language to do more traditional type programming especially string manipulation which is completely missing in GAMS. Moreover, the vast number of packages help solving scientific problems outside the scope of GAMS. In order to open this world to our customers in an easy way, GAMS comes with a Python 3 installation. By default, this installation is used in the Embedded Code Facility for Python and is ready to be used with the GAMS Python API. The Python installation is located in \[GAMS\text{ directory}/GMSPython\]. In contrast to earlier versions this installation comes without the Python package manager \texttt{pip} and therefore is not easily extendable. \texttt{GMSPython} comes with a selection of third party packages and modules that are either required by the APIs of the GAMS Python API collection or that are particularly helpful from within embedded Python code. Those packages are located in \[GAMS\text{ directory}/apifiles/Python/thirdparty\]. Running the \texttt{GMSPython} interpreter like follows will display all available modules:

\[\text{[GAMS directory]}/\text{GMSPython/python -c"help(\textquote{modules})"}\]

The list of redistributed third party packages is subject to constant changes and it is not recommended to rely on their availability. If you need packages that do not come with \texttt{GMSPython}, we recommend that you install your own version of Python and follow a few steps to connect GAMS with your Python installation. It is also possible to install \texttt{pip} for \texttt{GMSPython} using \texttt{get-pip}. While any Python installation should work in combination with GAMS we can recommend the \texttt{Anaconda} or \texttt{Miniconda} Python distributions. These Python distributions work with environments which allow you to have many different Python installations at the same time that don’t get in each other’s way.

During the lifetime of the GAMS process a single Python interpreter is used and the code "lives" in Python’s global scope. This means that objects defined in one embedded Python code section are still there in another embedded Python code section:

\begin{verbatim}
$onEmbeddedCode Python:
s = 'Hello world'
$offEmbeddedCode
$onEmbeddedCode Python:
gams.printLog(s)
$offEmbeddedCode
\end{verbatim}

will produce the following output in the GAMS log:

\begin{verbatim}
--- Starting compilation
--- main.gms(3) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- main.gms(6) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
Hello world
--- main.gms(7) 2 Mb
\end{verbatim}

If the GAMS process stops and restarts during execution, e.g. because of a solve with \texttt{solveLink=0}, the Python interpreter will be started new. This can cause complications if GAMS is in a sequence of \texttt{pause/continueEmbeddedCode} as discussed above.
4.45.5.1 Interface between GAMS and Python

The Python class `ECGamsDatabase` is the interface between GAMS and Python. An instance of this class is automatically created when an embedded code section is entered and can be accessed using the identifier `gams`. The following methods can be used in order to interact with GAMS:

```python
    gams.get(symbolName, keyType=KeyType.STRING, keyFormat=KeyFormat.AUTO, valueFormat=ValueFormat.AUTO, recordFormat=RecordFormat.AUTO)
```

This method retrieves an iterable object representing the symbol identified with `symbolName`. Typically there are two possibilities to access the records. Iterating using e.g. a `for` loop provides access to the individual records. By calling `list()` on the iterable object, a list containing all the data is created. Several optional parameters can be used in order to modify the format of the retrieved data:

- **keyType**: Determines the data type of the keys. It can be either `KeyType.STRING` (labels) or `KeyType.INT` (label indexes). The default setting is `KeyType.STRING`.
- **keyFormat**: Specifies the representation of the keys. Possible values are as follows:
  - `KeyFormat.TUPLE`: Encapsulate keys in a tuple
  - `KeyFormat.FLAT`: No encapsulation
  - `KeyFormat.SKIP`: Keys are skipped and do not appear in the retrieved data
  - `KeyFormat.AUTO` (default): Depending on the dimension of the GAMS symbol, a default format is applied:
    * Zero dimensional/scalar: `KeyFormat.SKIP`
    * One dimensional: `KeyFormat.FLAT`
    * Multi dimensional: `KeyFormat.TUPLE`
- **valueFormat**: Specifies the representation of the values. Possible values are as follows:
  - `ValueFormat.TUPLE`: Encapsulate values in a tuple
  - `ValueFormat.FLAT`: No encapsulation
  - `ValueFormat.SKIP`: Values are skipped and do not appear in the retrieved data
  - `ValueFormat.AUTO` (default): Depending on the type of the GAMS symbol, a default formats is applied:
    * Set: `ValueFormat.SKIP`
    * Parameter: `ValueFormat.FLAT`
    * Variable/Equation: `ValueFormat.TUPLE`
- **recordFormat** Specifies the encapsulation of records into tuples. Possible values are as follows:
  - `RecordFormat.TUPLE`: Encapsulates every record in a tuple
  - `RecordFormat.FLAT`: No encapsulation. Throws an exception if it can not be applied. It is guaranteed that the length of a retrieved Python list is equal to the number of records of the corresponding GAMS symbol. This principle leads to an incompatibility of `RecordFormat.FLAT` whenever a record consists of more than one item (e.g. multi dimensional symbols, variables and equations which have five numeric values).
  - `RecordFormat.AUTO` (default): Depending on the number of items that represent a record, a default format is applied. If possible this is always `RecordFormat.FLAT`.

GAMS special values `NA`, `INF`, and `-INF` will be mapped to IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')`. GAMS special value `EPS` will be either mapped to 0 or to the small numeric value 4.94066E-324 depending on the setting of flag `gams.epsAsZero`.

The following Python code shows some examples of `gams.get` and illustrates the use of different formats:
Set i / i1 text 1, i2 text 2 /
  j / j1*j2 /

Scalar p0 /3.14/;
Parameter p1(i) / #i 3.14 /
  p2(i,j) / i1.#j 3.14 /
Variable v0 / fx 3.14 /;
equation e1(i) / #i.fx 3.14 /
  e2(i,j) / i1.#j.fx 3.14 /;

$onEmbeddedCode Python:
# scalar parameter
gams.printLog(f"{list(gams.get('p0'))}")               # prints [3.14]
gams.printLog(f"{list(gams.get('p0', recordFormat=RecordFormat.TUPLE))}") # prints [(3.14,)]

# one dimensional parameters:
gams.printLog(f"{list(gams.get('p1'))}")           # prints [('i1', 3.14)]
gams.printLog(f"{list(gams.get('p1', keyFormat=KeyFormat.TUPLE))}") # prints [('i1', (3.14,))]
gams.printLog(f"{list(gams.get('p1', valueFormat=ValueFormat.TUPLE))}") # prints [('i1', (3.14,))]
gams.printLog(f"{list(gams.get('p1', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE))}") # prints [('i1', (3.14,))]

# two dimensional parameter:
gams.printLog(f"{list(gams.get('p2'))}")       # prints [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)]
gams.printLog(f"{list(gams.get('p2', keyFormat=KeyFormat.FLAT))}") # prints [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)]

# one dimensional sets:
gams.printLog(f"{list(gams.get('i'))}")      # prints ['i1', 'i2']
gams.printLog(f"{list(gams.get('i', valueFormat=ValueFormat.FLAT))}") # prints [('i1', 'text 1'), ('i2', 'text 2')]

# scalar variables/equations
gams.printLog(f"{list(gams.get('v0'))}")      # prints [(3.14, 0.0, 3.14, 3.14, 1.0)]

# one dimensional variables/equations:
gams.printLog(f"{list(gams.get('e1'))}")       # prints [('i1', (3.14, 0.0, 3.14, 3.14, 1.0))]
gams.printLog(f"{list(gams.get('e1', valueFormat=ValueFormat.FLAT))}") # prints [('i1', (3.14, 0.0, 3.14, 3.14, 1.0))]

# two dimensional variables/equations:
gams.printLog(f"{list(gams.get('e2'))}")       # prints [('i1', 'j1', 3.14, 0.0, 3.14, 3.14, 1.0), ('i1', 'j2', 3.14, 0.0, 3.14, 3.14, 1.0)]

gams.printLog(f"{list(gams.get('e2', keyFormat=KeyFormat.FLAT, valueFormat=ValueFormat.FLAT))}") # prints [('i1', 'j1', 3.14, 0.0, 3.14, 3.14, 1.0), ('i1', 'j2', 3.14, 0.0, 3.14, 3.14, 1.0)]

# using label indexes instead of labels

gams.printLog(f"{list(gams.get('p1', keyType=KeyType.INT))}") # prints [(1, 3.14), (2, 3.14)]
gams.printLog(f"{list(gams.get('i', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE, keyType=KeyType.INT))}") # prints [(1, ('text 1',)), (2, ('text 2',))]

$offEmbeddedCode

gams.set(symbolName, data, mergeType=MergeType.DEFAULT, domCheck=DomainCheckType.DEFAULT, mapKeys=mapKeys)

This method sets the data for the GAMS symbol identified with symbolName. The parameter data takes a Python list or set containing items that represent the records of the symbol. It is also possible to pass an instance of a subclass of _GamsSymbol (e.g. GamsParameter or GamsSet) when using the Object-oriented GAMS Python API in an embedded code section. In case of a Python list or set, depending on the type and the dimension of the symbol, different formats can be used in order to specify the data. Different formats cannot be mixed within one list. In general each record needs to be represented as a tuple containing the keys and the value field(s). Keys and/or values can also be enclosed in a tuple. Keys can be entered as labels (string) or label indexes (int). The argument mapKeys...
allows to pass a callable to remap the elements of the key (e.g. turn them explicitly into strings via `mapKeys=str`). Value fields depend on the type of the symbol:

- Parameters: One numerical value
- Sets: explanatory text (optional)
- Variable/Equations: Five numerical values: level, marginal, lower bound, upper bound, scale/prior/stage

IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')` will be remapped to GAMS special values `NA`, `INF`, and `-INF`. The small numeric value `4.94066E-324` will be mapped into GAMS special value `EPS`.

The following Python code gives some examples on different valid formats for different symbol types and dimensions:

```python
Set i / i1 text 1, i2 text 2 /
  j / j1*j2 /
i(i);
Scalar p0;
Parameter p1(i)
p2(i,j);
equation e1(i)
e2(i,j);
$onEmbeddedCode Python:
# scalar parameter
gams.set('p0', [3.14])
gams.set('p0', [(3.14,)])
# one dimensional parameters:
gams.set('p1', [('i1', 3.14), ('i2', 3.14)])
gams.set('p1', [('i1',), (3.14,)], [('i2',), (3.14,)])
gams.set('p1', [('i1', (3.14,)), ('i2', (3.14,))])
gams.set('p1', [('i1',), (3.14,)], [('i2',), (3.14,)])
# two dimensional parameter:
gams.set('p2', [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)])
gams.set('p2', [('i1', 'j1'), (3.14,)], [('i1', 'j2'), (3.14,)])
# one dimensional sets:
gams.set('ii', ['i1', 'i2'])
gams.set('ii', [('i1',), ('i2',)])
# one dimensional sets with explanatory text
gams.set('ii', [('i1', 'text 1'), ('i2', 'text 2')])
gams.set('ii', [('i1',), ('text 1'), ('i2',), ('text 2')])
# one dimensional variables/equations:
gams.set('e1', [('i1', 3.14, 0, 0, 10, 1), ('i2', 3.14, 0, 0, 10, 1)])
gams.set('e1', [('i1', (3.14, 0, 0, 10, 1)), ('i2', (3.14, 0, 0, 10, 1))])
gams.set('e1', [('i1',), (3.14, 0, 0, 10, 1)], [('i2',), (3.14, 0, 0, 10, 1)])
# two dimensional variables/equations:
gams.set('e2', [('i1', 'j1', 3.14, 0, 0, 10, 1), ('i1', 'j2', 3.14, 0, 0, 10, 1)])
gams.set('e2', [('i1', 'j1'), (3.14, 0, 0, 10, 1)], [('i1', 'j2'), (3.14, 0, 0, 10, 1)])
# using label indexes instead of labels
gams.set('p1', [(1, (3.14,)), (2, (3.14,))]) # one dimensional parameter
gams.set('ii', [(1, 'text 1'), (2, 'text 2')]) # one dimensional set
gams.set('e2', [(1, 3, (3.14, 0, 0, 10, 1)), ((4, (3.14, 0, 0, 10, 1)))] # two dimensional variables/equations
$offEmbeddedCode p0 p1 p2 ii e1 e2
The optional parameter mergeType specifies if data in a GAMS symbol is merged (MergeType.MERGE) or replaced (MergeType.REPLACE). If left at MergeType.DEFAULT it depends on the setting of $on/offMulti[R] if GAMS does a merge, replace, or trigger an error during compile time. During execution time MergeType.DEFAULT is the same as MergeType.MERGE.

The optional parameter domCheck specifies if Domain Checking is applied (DomainCheckType.CHECKED) or if records that would cause a domain violation are filtered (DomainCheckType.FILTERED). If left at DomainCheckType.DEFAULT it depends on the setting of $on/offFiltered if GAMS does a filtered load or checks the domains during compile time. During execution time DomainCheckType.DEFAULT is the same as DomainCheckType.FILTERED.

Note

When calling gams.set() in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

\[ \text{gams.getUel(idx)} \]

Returns the label corresponding to the label index idx

\[ \text{gams.mergeUel(label)} \]

Adds label to the GAMS universe if it was unknown and returns the corresponding label index.

Note

When calling gams.mergeUel() in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

\[ \text{gams.getUelCount()} \]

Returns the number of labels.

\[ \text{gams.printLog(msg)} \]

Print msg to log.

\[ \text{gams.arguments} \]

Contains the command line that was passed to the Python interpreter of the embedded code section. The syntax for passing arguments to the Python interpreter can be seen above.

\[ \text{gams.epsAsZero} \]

Flag to read GAMS EPS as 0 (True) or as a small number, \(4.94066 \times 10^{-324}\), when set to False. Default is True.

\[ \text{gams.ws} \]

Property to retrieve an instance of GamsWorkspace that allows to use the Object-oriented GAMS Python API. The instance is created when the property is read for the first time using a temporary working directory. A different working directory can be specified using gams.wsWorkingDir. For debug output, the property gams.debug can be set to a value from DebugLevel.
4.45 Embedded Code Facility

**gams.wsWorkingDir**

Property that can be specified before accessing `gams.ws` for the first time in order to set the working directory. Setting the property after the first call to `gams.ws` will have no effect on the created GamsWorkspace.

**gams.db**

Property to retrieve an instance of `GamsDatabase`. The instance is created when the property is read for the first time and allows to access the GAMS symbols using the methods of the Object-oriented GAMS Python API.

**gams.debug**

Property that can be set to a value from `DebugLevel` for debug output. Default is `DebugLevel.Off` (0). Setting this property affects both the debug output from embedded code and the debug output from the Object-oriented API. The property needs to be changed before accessing `gams.ws` for the first time in order to take effect in the Object-oriented API. Setting the property after the first call to `gams.ws` will have no effect on the GamsWorkspace.

### 4.45.5.2 Using the Object-oriented API

The EGCamsDatabase class provides mechanisms for using the Object-oriented GAMS Python API in an embedded code section. The property `gams.ws` can be used to get an instance of GamsWorkspace. The property `gams.db` allows to access an instance of GamsDatabase that can be used to read and write data from the internal GAMS database like it can be done using `gams.get` and `gams.set` but using the access mechanisms of the GamsDatabase class. In addition to the different reading and writing mechanisms offered by the OO-API other classes like `GamsModelInstance` can be used inside embedded Python code. This combination provides programmatic access to a generated model instance and exchange of update parameters and results between Python and GAMS. The GAMS Library model `embmiex1` gives an example of how to combine embedded Python code and a `GamsModelInstance` which utilized the library include `$libInclude pyEmbMI`.

Moreover, the GAMS Transfer Python API can be combine with the OO-API in the embedded code context as the following example demonstrates:

```python
Set i / i1*i1000 /; alias(i,j);
Parameter a(i,j), aInv(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;

import gamstransfer as gt
import numpy as np

m = gt.Container(gams.db)
A = m.data["a"].toDense()
m.data["aInv"].setRecords(np.linalg.inv(A))
m.write(gams.db, ["aInv"]) endEmbeddedCode aInv
```
4.45.5.3 Exchange via Files and Environment Variables

The Python class ECGamsDatabase provides read and write access to GAMS symbols. There are two other communication methods that can be used at GAMS compile time: files and environment variables. At compile time the Python code can produce a text file that can be included into GAMS via $include as in the following example:

```python
$onEmbeddedCode Python: 10
with open('i.txt', 'w') as f:
    for i in range(int(gams.arguments)):
        f.write(f'i{i+1}\n')
$offEmbeddedCode

Set i /
$include i.txt
/;
display i;
```

Here the Python code received the number of elements to write to a text file via the argument after Python:. This text file is then included in the data statement of the GAMS set i. The display in the listing file looks as follows:

```plaintext
---- 20 SET i
i1 , i2 , i3 , i4 , i5 , i6 , i7 , i8 , i9 , i10
```

Python provides many packages to read input files for many different formats and hence can be used to transform such formats to a GAMS compatible input format, as an alternative to providing the data via list objects and the gams.set functionality.

The second alternative to exchange information at compile time are environment variables. GAMS and Python allow to get and set environment variables and hence can be conveniently used to exchange small pieces of information. The following code provides an example where the maximum value of a parameter b is needed to build a set k:

```gams
Set i / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;
$onEmbeddedCode Python:
import os
os.environ["MAXB"] = str(int(max([b for i,b in gams.get("b")])))
gams.printLog(f'max value in b is {os.environ["MAXB"]}')
$offEmbeddedCode

$if x%sysEnv.MAXB%=x $abort MAXB is not set
Set k "from 0 to max(b)" / 0*%sysEnv.MAXB% /;
$eval CARDK card(k)
$log card(k)=%CARDK%
```

It is important to understand that os.environ is initialized when Python imports the os module. Hence, if we later set an environment variable in GAMS (via $setEnv ABC abc or put_utility "setEnv" / "ABC" / "abc") and access this in some embedded code via os.environ["ABC"], this will not provide the expected value "abc". In such a case the environment variable needs to be accessed via gams.get_env("ABC") (and can be stored in os.environ["ABC"]) as demonstrated by the following code:
4.45 Embedded Code Facility

$onEmbeddedCode Python:
import os
os.environ["TEST"] = 'Hello world'
$offEmbeddedCode
$log TEST=%sysEnv.TEST%
$setEnv ABC abc
$onEmbeddedCode Python:
gams.printLog(f'ABC={os.environ.get("ABC", "ABC not set")}')
gams.printLog(f'ABC={gams.get_env("ABC")}')
$offEmbeddedCode

which produces the following output:

--- Starting compilation
--- main.gms(4) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
TEST=Hello world
--- main.gms(10) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
ABC=ABC not set
ABC=abc

Obviously, the set \( k \) can also be filled directly with data from within Python:

Set   i / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;
Set   k(*) "from 0 to max(b)";

$onEmbeddedCode Python:
kmax = max([b for i,b in gams.get("b")])
gams.set('k',[str(k) for k in range(int(kmax+1))])
$offEmbeddedCode

In both cases the resulting GAMS symbol \( k \) is a set with elements 0 to 59.

Note that the \((*)\) in the declaration of \( k \) is necessary to inform the compiler of dimensionality of the symbol \( k \). GAMS symbol declarations can be done without domain list, so a set \( k \) does not necessarily mean that this is a one dimensional set. The compiler will learn the dimensionality from the use of \( k \). For example, if \( k \) shows up as a domain set in a declaration (parameter \( p(k) \)) it is clear that \( k \) is a one dimensional set. Without knowing the dimensionality \( k \) will not be available with embedded code or exported (at compile-time) to GDX and the following code

Set k;
$onEmbeddedCode Python:
gams.set('k', ['0','1','2'])
$offEmbeddedCode

will fail with compilation error Cannot load/unload symbol with unknown dimension.
4.45.5.4 Encodings

Python 3 expects source code to be encoded in UTF-8 per default. Therefore all embedded Python code has to be encoded in UTF-8 as well. However, this can be customized by adding a comment in the format # coding=<encoding name> or # -*- coding: <encoding name> -*- as first line of an Embedded Python Code section. The encoding can be changed from any Embedded Python Code section and will affect all subsequent sections. So in case of a GAMS source file which is encoded using a non-UTF-8 encoding like cp1252 or others, it is sufficient to change the encoding only once in the very first Embedded Code section. Note that UTF-16 encoding is not supported.

Note
Changing the source code encoding has no effect on the default encoding that is used by the open() function since it uses whatever locale.getpreferredencoding(False) returns. Furthermore the encoding used for stdin/stdout/stderr has to be controlled separately. This can be achieved by setting the environment variable PYTHONIOENCODING.

4.45.5.5 Multiple Independent Python Sessions

At execution time the user has the ability to pause and continue an embedded code segment. As explained at the beginning, the user Python code lives in Python's global scope. Nevertheless, besides some performance aspects one can also allows work with multiple independent Python sessions. Due to some Python module incompatibilities (e.g. numpy) the independent Python sessions have to be enabled with a command line option pyMultInst set to 1. After the pauseEmbeddedCode we can obtain and store the handle of the last embedded code execution via function embeddedHandle. The handle needs to be supplied when we continue the Python session via continueEmbeddedCode. The different Python sessions are fairly separate as shown in the example below. Here we save the GAMS scalar ord(i) in a Python object i five times. The value for i that we store in the five different Python sessions is 1 to 5. In the subsequent loop we activate the Python session with the appropriate handle and print the value of i:

```gams
$if not %sysEnv.GMSPYTHONMULTINST%=1 $abort.noError Start with command line option pyMultInst=1
Set i / i1*i5 /;
Parameter h(i) 'session handle';
loop(i, 
   put_utility 'setEnv' / 'ORDI' / ord(i):0:0;
   embeddedCode Python: 
   i = int(gams.get_env("ORDI"))
   gams.printLog(f'\{i}\n')
   pauseEmbeddedCode
   h(i) = embeddedHandle;
);
loop(i, 
   continueEmbeddedCode h(i):
   gams.printLog(f'\{i}\n')
   endEmbeddedCode
);
```

The GAMS log shows the value of i in the different Python sessions:

```
--- Starting execution: elapsed 0:00:00.002
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
1
--- Initialize embedded library embpycclib64.dll
```
4.45 Embedded Code Facility

--- Execute embedded library embpycclib64.dll
2
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
3
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
4
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
5
--- Execute embedded library embpycclib64.dll
1
--- Execute embedded library embpycclib64.dll
2
--- Execute embedded library embpycclib64.dll
3
--- Execute embedded library embpycclib64.dll
4
--- Execute embedded library embpycclib64.dll
5
*** Status: Normal completion

4.45.5.6 Troubleshooting Embedded Python Code

The GAMS compiler ensures that the number of errors during execution time is minimized. While the logic of the GAMS program might be flawed there is nothing (with a few exceptions) that the GAMS system cannot execute. This is different if we embed foreign code in a GAMS program. The GAMS compiler does not understand the foreign code syntax and just skips over it. Only when the code is executed we will find out if everything works as expected. If the embedded code contains some (Python) syntax errors the Python parser will inform us about this and the message will appear in the GAMS log. For example, the following Python code using the gams.printLog function two times will generate a syntax error:

```plaintext
$onEmbeddedCode Python:
gams.printLog('hello')
gams.printLog('world...')
$offEmbeddedCode
```

The GAMS log will provide some guidance:

```plaintext
--- Execute embedded library libembpycclib64.so
File "<string>", line 7
  gams.printLog('world...')

IndentationError: unexpected indent
Python error! Return code from Python execution: -1
*** Error executing embedded code section:
*** Check log above
```

Moreover, if the Python code raises an exception which is not handled within the code this will also lead to a compilation or execution error in GAMS depending at what phase the embedded code is executed.
embeddedCode Python:
raise Exception('something is wrong')
endEmbeddedCode

will produce the following GAMS log and an execution time error:

--- Initialize embedded library libembpycclib64.so
--- Execute embedded library libembpycclib64.so
Exception from Python (line 0): something is wrong
*** Error at line 1: Error executing "embeddedCode" section: Check log above

Note
It is good practice to raise a Python exception if an error occurs. In any case using `exit()` needs to be avoided since it terminates the executable in an uncontrolled way.

The Python code is executed as part of the GAMS process and GAMS gives control to the Python interpreter when executing the embedded code. So in the worst case if the Python interpreter crashes, the entire GAMS process will crash. Therefore, it is important to be able to test and debug the embedded Python code independent of GAMS. In the following examples we call the Python interpreter executable as part of a GAMS job. In principle this can be tested and debugged completely independent of GAMS where a GDX file represents the content of the GAMS database.

In the first example we mimic the embedded code facility at compile time by exporting the entire GAMS database to a GDX file `debug.gdx`. With `$on/offEcho` we write the embedded code with a few extra lines at the top and bottom surrounded by a `try/except` block and execute the Python interpreter via `$call`. One of the extra lines at the end of the embedded code triggers the creation of a GDX result file `debugOut.gdx` which can be imported in subsequent `$gdxin/$load` commands.

```plaintext
Set i /i1*i10/
   p(i,i) permutation;

$gdxOut debug.gdx
$unload
$gdxOut
$onEcho > debug.py
from gamsemb import *
gams = ECGAMSDatabase(r'\%gams.sysdir%\[:-2\], 'debug.gdx')
try:
   import random
   i = list(gams.get("i"))
   p = list(i)
   random.shuffle(p)
   for idx in range(len(i)):
      p[idx] = (i[idx], p[idx])
   gams.set("p", p)
gmdWriteGDX(gams._gmd,'debugOut.gdx',1);
except Exception as e:
   print(str(e))
$offEcho
$call.checkErrorLevel ="%gams.sysdir%GMSPython/python" debug.py

$gdxIn debugOut.gdx
$loadDC p

Option p:0:0:1;
Display p;
```
In the second example we mimic the embedded code facility at execution time by exporting the entire GAMS database to a GDX file `debug.gdx` via `execute_unload 'debug.gdx';`. We write the embedded code with a few extra lines at the top and bottom surrounded by a try/except block via the `put` facility and execute the Python interpreter via `execute`. Identical to the compile time example, we export the result to the GDX file `debugOut.gdx` which can be imported via the `execute_load` statement.

```plaintext
Set i /i1*i10/
   p(i,i) permutation;
execute_unload 'debug.gdx';
file fpy / 'debug.py' /; put fpy;

$onPutS
from gamsemb import *
gams = ECGAMSDatabase(r'\%gams.sysdir\ %[:2],\'debug.gdx\')
try:
    import random
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
    gmdWriteGDX(gams._gmd,'debugOut.gdx',1);
except Exception as e:
    print(str(e))
$offPut
putClose fpy;
execute.checkErrorLevel ="%gams.sysdir%GMSPython/python" debug.py';

execute_load 'debugOut.gdx', p;
Option p:0:0:1;
Display p;
```

### 4.45.5.6.1 Automatic Indentation

All embedded Python code is automatically wrapped in a `try-except` block that requires an indentation (two spaces). While this additional indentation makes no difference for most Python code, it does make a difference when it comes to multi-line strings. Therefore, multi-line strings created with triple quotes (either single or double quotes) are excluded from this rule to avoid unintentional spaces, e.g. when writing to a file:

```plaintext
$onEmbeddedCode Python:
f = open("file.txt", "w")
f.write('''some
 multi
 line
 string
''')
f.close()
$offEmbeddedCode
```

Still, when using the line continuation character (`\`) in combination with strings, one has to be aware of the extra spaces that are added automatically:
This will print *some multi line string* instead of *somemultilinestring* due to the automatically applied indentation.

### 4.45.5.7 Performance Considerations of Embedded Python Code

If the same embedded code section (e.g. in a loop) is executed many times there are a few considerations to be taken into account in order to get the best performance. For this we will experiment with the example from the introduction. We look for a random permutation of a set `i`. In addition we have a cost matrix `c(i,ii)` and we are looking for the least cost permutation. We should just formulate this as matching in a bi-partite graph but in order to demonstrate some performance considerations we will repeatedly call the Python code that provides a random permutation and we will evaluate the cost of the permutation in GAMS and store the value of the cheapest one. Here is the naive implementation using embedded code:

```python
import random
delattr(gams, 'p')
gams.printLog(s)
```

Even though the Python interpreter stays alive the fresh start and end of an embedded code section goes together with the setup and initialization of multiple data structure, including the `gams` object of type `ECGAMSDatabase`. We can avoid the repeated setup and initialization by using `pause` and `continue`:

```python
import random
gams.set('p', [ (i[idx], p[idx]) for idx in range(len(i)) ])
```
embeddedCode Python:
import random
pauseEmbeddedCode

Set iter / 1*100 /
Scalar tcost
    minTCost / +inf /;
loop(iter,
    continueEmbeddedCode:
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    gams.set("p", [ (i[idx], p[idx]) for idx in range(len(i)) ])
    pauseEmbeddedCode p
tcost = sum(p, c(p));
    if (tcost < minTCost, minTCost = tcost);
); continueEmbeddedCode:
    pass
endEmbeddedCode
Display minTCost;

The last embedded code execution of the Python pass statement is to clean up and terminate the Python session. Since the amount of data is relatively small, there is little difference in the running time between these variants. The performance impact of independent Python session (command line option pyMultInst) is significant. While the pause/continue code runs at the same speed (roughly 0.2 seconds) independent of the pyMultInst setting, the original code runs much slower (about 6.6 seconds) because every iteration results in a newly created session.

There are other performance considerations. We can actually extract the set i once before the GAMS loop starts. Moreover, we can work with label indexes rather than the labels itself. Indexes are integers and are often faster than labels that are stored as strings. The only difference to the code above is the extraction method of the Python list i by i = list(gams.get("i",keyType=KeyType.INT)).

### 4.45.5.8 Extending GMSPython

While we recommend to use your own Python installation if you need additional packages, there are ways to extend the Python system that comes with GAMS in GMSPython. Here are the steps:

2. Update pip
3. Install additional packages

We highly recommend installing pip and additional packages in the user site (use --user when running get-pip and pip). This is especially important for macOS users. Installing files in the GAMS system directory may interfere with the file notarization and may prevent GAMS from starting properly. Here is a typical dialog for Linux/macOS using curl to download get-pip.py. If you don't have curl or wget, use your web browser for downloading.

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
/path/to/gams/GMSPython/python get-pip.py --user
"/local/bin/pip install -U pip
"/local/bin/pip install geocoder --user
```
4.45.5.9 Using an External Python 3 Installation

Let's assume you don't want to work with the GAMS distributed GMSPython but you want to connect GAMS with a Python 3 installation of your liking. In order for the Embedded Code Facility to work, you need to make the alternative Python interpreter aware of the required modules provided by GAMS by following the Getting started steps in the Python API tutorial.

In addition to this you need to do one more step to allow GAMS to find your Python installation when executing Embedded Python code: you need to point to the Python dynamic load library or shared object. On Windows this file is called python3X.dll, on Linux libpython3.X.so, and on macOS libpython3.X.dylib. The X stands for the minor version number of Python 3. While embedded code has been built and tested for Python 3.8, chances are that you can also point to other Python 3 libraries. Since the expert level API is required, the set of versions is limited to what GAMS supports (currently GAMS ships these for Python \( \geq 3.6 \)). Limited experiments with the embedded code library for Python 3.8 with Python 3.6, 3.7, and 3.9 libraries were successful. Python 3.10 is not supported yet. You need to set the environment variable GMSPYTHONLIB to point GAMS to the Python library:

- **Windows:**
  ```bash
  set GMSPYTHONLIB=c:\path\to\python\python38.dll
  ```

- **Linux:**
  ```bash
  export GMSPYTHONLIB=/path/to/python/lib/libpython3.8.so
  ```

- **Mac OSX:**
  ```bash
  export GMSPYTHONLIB=/path/to/python/lib/libpython3.8.dylib
  ```

GAMS will extract the Python home from the location of the Python library. In case this extraction does not work (because of an unconventional Python installation), you can instruct GAMS to set the Python home via the additional environment variable GMSPYTHONHOME. Please note, that this environment variable only needs to be set for GAMS Embedded Python code, so it is sufficient to add these variables to the GAMS configuration YAML file.

**Note**
When using environment-based Python distributions like Anaconda or Miniconda, it is not sufficient to set GMSPYTHONLIB (and GMSPYTHONHOME). The environment to be used needs to be activated properly before (e.g. `conda activate myEnv`). Packages like numpy which have been installed using conda (`conda install numpy`) won't work otherwise. In addition to setting GMSPYTHONLIB we therefore recommend to start GAMS itself and also tool like GAMS Studio from an activated conda environment in order to make sure that the environment and all installed packages work as expected.

Although it is recommended to use a self-contained Python installation, it is possible to use a virtual environment instead (e.g. via `venv`). In this case, GMSPYTHONLIB has to be set to the Python library of the installation from which the environment was derived from. In addition, the PYTHONPATH environment variable needs to be set to the site-packages directory of the virtual environment in order to make its packages available. Note that with this setup the site-packages directory of the original Python installation will also be found by GAMS Embedded Python code.

4.45.5.10 Building your own Embedded Python Code Library

Although the Embedded Code Facility and its binary components are part of the GAMS distribution, it is possible to build it manually from source using the following commands. The exact command line might change depending on the compiler and the operating system in use:

- **Windows:**
  ```bash
  ```

- **Linux:**
  ```bash
  ```

- **Mac OSX:**
  ```bash
  ```

Although it is recommended to use a self-contained Python installation, it is possible to use a virtual environment instead (e.g. via `venv`). In this case, GMSPYTHONLIB has to be set to the Python library of the installation from which the environment was derived from. In addition, the PYTHONPATH environment variable needs to be set to the site-packages directory of the virtual environment in order to make its packages available. Note that with this setup the site-packages directory of the original Python installation will also be found by GAMS Embedded Python code.
4.45 Embedded Code Facility

- **Linux:**
  ```bash
cd [GAMS directory]/apifiles/C/api
icl.exe -Feembpycclib64.dll -IC:\path\to\python\include embpyoo.c gmdcc.c gclgms.c -LD -link -nodefaultlib:python38.lib
  ```
- **Mac OS X:**
  ```bash
cd [GAMS directory]/apifiles/C/api
gcc -o libembpycclib64.dylib -dynamiclib -shared -m64 -install_name @rpath/libembpycclib64.dylib embpyoo.c gmdcc.c gclgms.c -lm -ldl
  ```

4.45.6 Connect

The Connect framework gives unified and platform independent access to data exchange with different formats (e.g. CSV and Excel). The instructions how to access the various data sources are given in YAML syntax. These YAML instructions are the code for the embedded Connect code. The embedded Connect code engine will parse the YAML instructions and use Connect agents to act upon the instructions. The agents GAMSReader and GAMSWriter send data back and forth to GAMS. A simple example is given above.

Embedded code Connect is build on top of embedded code Python. Similar considerations (e.g. Python interpreter is loaded once) carry over from Python to Connect. Moreover, the agent PythonCode has access to the object `gams` of type `ECGamsDatabase` which is used for communication of data with GAMS. In contrast to embedded Python code, the code of the agent PythonCode does not execute in the global scope but in a separate block. This means that variables set in the code are not available in another PythonCode section. The following code ends up in the except clause and prints `Cannot access my_i`.

```plaintext
eMBEDDED_CODE
- **PythonCode:**
  ```python
code: |
  my_i = 1
  ```
- **PythonCode:**
  ```python
code: |
  try:
      gams.printLog(f'{my_i}')
  except:
      gams.printLog('Cannot access my_i')
```

END EMBEDDED_CODE

Since Python allows to add attributes to an object at runtime, the `my_i` variable can be added to the `gams` object and hence becomes available in different PythonCode sections and even in different embedded Connect code parts (when pause/continue is used):

```plaintext
eMBEDDED_CODE
- **PythonCode:**
  ```python
code: |
  gams.my_i = 1
  ```
- **PythonCode:**
  ```python
code: |
  gams.printLog(f'{gams.my_i}')
```

PAUSE EMBEDDED_CODE

CONTINUE EMBEDDED_CODE:

- **PythonCode:**
  ```python
code: |
  gams.printLog(f'{gams.my_i}')
```

END EMBEDDED_CODE

Even though embedded code Python and Connect share the same basis, they run in different instantiations of the Python interpreter and hence don't share e.g. the same gams object.
4.45.6.1 Substitutions in Embedded Connect Code

The GAMS compiler has mechanisms to substitute compile-time variables to parameterize GAMS code including embedded code (unless the V, e.g. `$onEmbeddedCodeV`, of embedded code is used). Here is an example how to use double-dash parameters to allow the user to set the CSV file name via the GAMS command line (gams mymodel.gms --csvfilename=mystockprice.csv):

```
$if not set CSVFILENAME $set CSVFILENAME stockprice.csv
$onEmbeddedCode
Connect:
  CSVReader:
  - file: %CSVFILENAME%
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
  GAMSWriter:
    symbols:
      - name: stockprice
$offEmbeddedCode
```

Not only user defined compile-time variables will be substituted, but also all other compile time variables, e.g. `%gams.scrDir%` or `%system.dirSep%`. Another way to pass substitution instructions down to embedded Connect code are the embedded code arguments:

```
$if not set MYCSVFILENAME $set MYCSVFILENAME stockprice.csv
$onEmbeddedCodeV
Connect: --CSVFILENAME=%MYCSVFILENAME%
  CSVReader:
  - file: %CSVFILENAME%
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
  GAMSWriter:
    symbols:
      - name: stockprice
$offEmbeddedCode
```

Together with the put_utility command ECArguments that appsends some arbitrary text to the embedded code arguments this can be used to parameterize the embedded Connect code at execution time, e.g. to create files based on label names:

```
Set i /1*5/; Parameter p(i);
loop(i,
  p(i) = uniform(0,1);
  put_utility 'ECArguments' / '--CSVFILENAME=' i.tl:0 '.csv';
  EmbeddedCodeV
Connect:
  GAMSReader:
    symbols:
      - name: p
  CSVWriter:
    name: p
    file: %CSVFILENAME%
endEmbeddedCode
);
```

Please note that while the GAMS command line parameters allow various formats to specify double-dash parameters, the embedded code Connect substitution arguments must be in the form --key=value, so a = character to separate key and value with no spaces. The value might be a (double) quoted string though. Please also note that the embedded code arguments set with put_utility command ECArguments will be applied to any subsequent embedded code section. So one might want to clear the argument by adding a put_utility "ECArguments" / ""; after the relevant embedded code section.
4.46  GAMS Connect

4.46.1  Concept

GAMS Connect is a framework inspired by the concept of a so-called ETL (extract, transform, load) procedure that allows to integrate data from various data sources. The GAMS Connect framework consists of the Connect database and the Connect agents that operate on the Connect database. Via the available Connect interfaces the user passes instructions to call Connect agents for reading data from various file types into the Connect database, transforming data in the Connect database, and writing data from the Connect database to various file types. Instructions are passed in YAML syntax. Note that in contrast to a typical ETL procedure, read, transform and write operations do not need to be strictly separated.

4.46.2  Usage

GAMS Connect is available via the GAMS command line parameters ConnectIn and ConnectOut, via embedded code Connect, and as a standalone command line utility gamsconnect.

Instructions processed by the GAMS Connect interfaces need to be passed in YAML syntax as follows:

```
- <agent name1>:
  <global option1>: <value>
  <global option2>: <value>
  ...
  symbols:
    - <symbol1 option1>: <value>
      <symbol1 option2>: <value>
      ...
    - <symbol2 option1>: <value>
      <symbol2 option2>: <value>
      ...
- <agent name2>:
  ...
```

The user lists the tasks to be performed successively. All tasks begin at the same indentation level starting with a - (a dash and a space) followed by the Connect agent name and a : (a colon). Connect agent options are represented in a simple <option>: <value> form. Please check the documentation of Connect Agents for available options. Some agents can operate on multiple symbols. For these agents there are two types of options:

1. Global: Global options allow to define general settings, e.g. the file name.

2. Symbol: Symbol options allow to define symbol specific settings, e.g. the name of the symbol. Symbol options are defined in a sub-list under the keyword symbols.
Some Connect agent options are of type global and symbol simultaneously, i.e. the user can define a global value for the option but may change the value for some symbols specifically. Note that YAML syntax also supports an abbreviated form for lists and dictionary, e.g. `symbols: [{<symbol1 option1>: <value>, <symbol1 option2>: <value>}, {<symbol2 option1>: <value>, <symbol2 option2>: <value>}]`.

Here is an example that uses embedded Connect code to process instructions:

```plaintext
$onecho > distance.csv
i;j;distance in miles
seattle;new-york;2,5
seattle;chicago;1,7
seattle;topeka;1,8
san-diego;new-york;2,5
san-diego;chicago;1,8
san-diego;topeka;1,4
$offecho

$onecho > capacity.csv
i,capacity in cases
seattle,350.0
san-diego,600.0
$offecho

Set i 'Suppliers', j 'Markets';
Parameter d(i,j) 'Distance', a(i) 'Capacity';
$onEmbeddedCode Connect:
  - CSVReader:
    file: distance.csv
    name: distance
    indexColumns: [1, 2]
    valueColumns: [3]
    fieldSeparator: ' ;'
    decimalSeparator: ','
  - CSVReader:
    file: capacity.csv
    name: capacity
    indexColumns: [1]
    valueColumns: [2]
  - GAMSWriter:
    symbols:
      - name: distance
        newName: d
      - name: capacity
        newName: a
$offEmbeddedCode
```

In this example, we are reading two CSV files `distance.csv` and `capacity.csv` using the `CSVReader`. Then we directly write to symbols in GAMS using the `GAMSWriter`.

Note that even though GAMS is case insensitive, GAMS Connect is case sensitive, i.e., YAML instructions are treated case sensitive. This also includes, e.g., indices in CSV files. Consider the following example where the index `j2` should be substituted by `ABC` when reading the CSV file `y.csv`:

```plaintext
$onecho > y.csv
i1,j1,2.5
```
i1,j2,1.7
i2,j1,1.8
i2,j2,1.4

$offecho

set i,j;
parameter p(i<,j<);

$onEmbeddedCode Connect:
- CSVReader:
  file: y.csv
  name: p
  indexColumns: [1,2]
  valueColumns: [3]
  header: false
  indexSubstitutions: { j2: ABC }
- GAMSWriter:
  writeAll: true
$offEmbeddedCode

display i,j,p;

Since the YAML instructions are treated case sensitive, the index J2 will not be substituted.

<table>
<thead>
<tr>
<th>j1</th>
<th>J2</th>
<th>ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>2.500</td>
<td>1.700</td>
</tr>
<tr>
<td>i2</td>
<td>1.800</td>
<td>1.400</td>
</tr>
</tbody>
</table>

All instructions provided to the Connect framework are read using UTF-8 encoding (utf-8-sig). This can be customized by adding a comment in the format # coding=<encoding name> or # -*- coding: <encoding name> -*- as first line in the YAML code. Note that UTF-16 encoding is not supported.

### 4.46.3 Connect Agents Summary

Current Connect agents support the following data source formats: CSV, Excel, GDX and SQL. The following Connect agents are available:

<table>
<thead>
<tr>
<th>Connect agent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSVReader</td>
<td>Allows reading a symbol from a specified CSV file into the Connect database.</td>
</tr>
<tr>
<td>CSVWriter</td>
<td>Allows writing a symbol in the Connect database to a specified CSV file.</td>
</tr>
<tr>
<td>DomainWriter</td>
<td>Allows rewriting the domain information of an existing Connect symbol.</td>
</tr>
<tr>
<td>GAMSReader</td>
<td>Allows reading symbols from the GAMS database into the Connect database.</td>
</tr>
<tr>
<td>GAMSWriter</td>
<td>Allows writing symbols in the Connect database to the GAMS database.</td>
</tr>
<tr>
<td>GDXReader</td>
<td>Allows reading symbols from a specified GDX file into the Connect database.</td>
</tr>
<tr>
<td>GDXWriter</td>
<td>Allows writing symbols in the Connect database to a specified GDX file.</td>
</tr>
<tr>
<td>Options</td>
<td>Allows to set more general options that can affect the Connect database and</td>
</tr>
<tr>
<td></td>
<td>other Connect agents.</td>
</tr>
<tr>
<td>PandasExcelReader</td>
<td>Allows reading symbols from a specified Excel file into the Connect database.</td>
</tr>
<tr>
<td>PandasExcelWriter</td>
<td>Allows writing symbols in the Connect database to a specified Excel file.</td>
</tr>
<tr>
<td>PandasSQLReader</td>
<td>Allows reading symbols from a specified SQL database into the Connect database.</td>
</tr>
<tr>
<td>PandasSQLWriter</td>
<td>Allows writing symbols in the Connect database to a specified SQL database.</td>
</tr>
</tbody>
</table>
### Connect agent Description

<table>
<thead>
<tr>
<th>Connect agent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>Allows index reordering and projection onto a reduced index space of a GAMS symbol.</td>
</tr>
<tr>
<td>PythonCode</td>
<td>Allows executing arbitrary Python code.</td>
</tr>
<tr>
<td>RawExcelReader</td>
<td>Allows reading unstructured data from a specified Excel file into the Connect database.</td>
</tr>
</tbody>
</table>

### 4.46.4 Getting Started

We introduce the basic functionalities of GAMS Connect agents on some examples.

#### 4.46.4.1 Simple Connect Example with Spreadsheets

The following example is part of GAMS Model Library as model *cta* and shows how to read and write Excel spreadsheets. Here is a code snippet of the first lines:

```plaintext
Set
   i 'rows'
   j 'columns'
   k 'planes';

Parameter
   dat(k<,i<,j<) 'unprotected data table'
   pro(k,i,j)   'information sensitive cells';

* extract data from Excel workbook
$onEmbeddedCode Connect:
  - PandasExcelReader:
    file: cox3.xlsx
    symbols:
      - name: dat
        range: Sheet1!A1
        rowDimension: 2
        columnNameDimension: 1
      - name: pro
        range: Sheet2!A1
        rowDimension: 2
        columnNameDimension: 1
  - GAMSWriter:
    writeAll: True
$offEmbeddedCode

[...]
```

It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data for the parameters are read from the Excel file *cox3.xlsx* using the Connect agent *PandasExcelReader*. The PandasExcelReader agent allows reading data for multiple symbols that are listed under the keyword *symbols*, here, parameter *dat* and *pro*. For each symbol, the symbol name is given by option *name* and the Excel range by option *range*. The option *rowDimension* defines that the first 2 columns of the data range will be used for the labels. In addition, the option *columnNameDimension* defines that the first row of the data range will be used for the labels. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent *GAMSWriter*. The GAMSWriter agent makes
the parameters \texttt{dat} and \texttt{pro} available outside the embedded Connect code. Note that the sets \(i, j\) and \(k\) are defined implicitly through parameter \texttt{dat}.

Finally, after solving the \texttt{cox3c} model with alternative solutions, Connect can be used to export results to Excel:

\[
\text{loop(l$((obj.l - best)/best <= 0.01),}
\]
\[
\text{ll(l) = yes;}
\]
\[
\text{binrep(s,l) = round(b.l(s));}
\]
\[
\text{binrep('','','Obj',l) = obj.l;}
\]
\[
\text{binrep('','','mSec',l) = cox3c.resUsd*1000;}
\]
\[
\text{binrep('','','nodes',l) = cox3c.nodUsd;}
\]
\[
\text{binrep('Comp','Cells','Adjusted',l) = sum((i,j,k)$not s(i,j,k)), 1$round(adjn.l(i,j,k) + adjp.l(i,j,k));}
\]
\[
\text{s\(olve\ cox3c \text{min} \ obj \text{ using \ mip;}\}
\]

This time, we need to use execution-time embedded Connect code. The Connect agent \texttt{GAMSReader} imports the reporting parameter \texttt{binrep} into the Connect database. With the Connect agent \texttt{PandasExcelWriter} we write the parameter into the \texttt{binrep} sheet of the Excel file \texttt{results.xlsx}.

### 4.46.4.2 Simple Connect Example with CSV files

The following example (a modified version of the \texttt{trnsport} model) shows how to read and write CSV files. The full example is part of DataLib as model \texttt{connect03}. Here is a code snippet of the first lines:

\[
\$\text{onEcho} > \text{distance.csv}
\]
\[
i,\text{new-york,chicago,topeka}
\]
\[
\text{seattle,2.5,1.7,1.8}
\]
\[
\text{san-diego,2.5,1.8,1.4}
\]
\[
\$\text{offEcho}
\]
\[
\$\text{onEcho} > \text{capacity.csv}
\]
\[
i,\text{capacity}
\]
\[
\text{seattle,350}
\]
\[
\text{san-diego,600}
\]
\[
\$\text{offEcho}
\]
\[
\$\text{onEcho} > \text{demand.csv}
\]
\[
j,\text{demand}
\]
\[
\text{new-york,325}
\]
\[
\text{chicago,300}
\]
\[
\text{topeka,275}
\]
\[
\$\text{offEcho}
\]
It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data for the parameters are read from CSV files using the Connect agent CSVReader. The CSVReader agent, for example, reads the CSV file distance.csv and creates the parameter d in the Connect database. The name of the parameter must be given by the option name. Column number 1 is specified as the first domain set using option indexColumns. The valueColumns option is used to specify the column numbers 2, 3 and 4 containing the values. Per default, the first row of the columns specified via valueColumns will be used as the second domain set. The symbolic constant lastCol can be used if the number of index or value columns is unknown. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent GAMSWriter. The GAMSWriter agent makes the parameters d, a and b available outside the embedded Connect code. Note that the sets i and j are defined implicitly through parameter d.

Finally, after solving the transport model, Connect can be used to export results to a CSV file:

Model transport / all /;
solve transport using lp minimizing z;

EmbeddedCode Connect:
  - GAMSReader:
    symbols:
      - name: x
  - Projection:
    name: x.l(i,j)
    newName: x_level(i,j)
  - CSVWriter:
    file: shipment_quantities.csv
    name: x_level
    unstack: True
endEmbeddedCode
This time, we need to use execution-time embedded Connect code. The Connect agent \texttt{GAMSReader}
imports variable $x$ into the Connect database. With the Connect agent \texttt{CSVWriter} we write the variable
level to the CSV file \texttt{shipment\_quantities.csv}:

\begin{verbatim}
i_0,new-york,chicago,topeka
seattle,50.0,300.0,0.0
san-diego,275.0,0.0,275.0
\end{verbatim}

Setting the option \texttt{unstack} to \texttt{True} allows to use the last dimension as the header row.

\subsection{Simple Connect Example for SQL}

The following example (a modified version of the \texttt{whouse} model) shows how to read from and write to a
SQL database (sqlite). The full example is part of DataLib as model \texttt{connect04}. Here is a code snippet
of the first lines:

\begin{verbatim}
[...]
Set t 'time in quarters';

Parameter
  price(t) 'selling price ($ per unit)'
  istock(t) 'initial stock   (units)';

Scalar
  storecost 'storage cost ($ per quarter per unit)'
  storecap 'stocking capacity of warehouse (units)';

$onEmbeddedCode Connect:
- PandasSQLReader:
  connection: "sqlite:///whouse.db"
  symbols:
    - name: t
      query: "SELECT * FROM timeTable;"
      type: set
    - name: price
      query: "SELECT * FROM priceTable;"
    - name: istock
      query: "SELECT * FROM iniStockTable;"
    - name: storecost
      query: "SELECT * FROM storeCostTable;"
    - name: storecap
      query: "SELECT * FROM storeCapTable;"
- GAMSWriter:
  writeAll: True
$offEmbeddedCode
[...]
\end{verbatim}

It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data
for all the symbols are read from the sqlite database \texttt{whouse.db} using the Connect agent \texttt{PandasSQLReader}
by passing the connection url through the option \texttt{connection}. The PandasSQLReader agent, for example,
queries the table \texttt{priceTable} for data and creates the parameter \texttt{price} in the Connect database. The
PandasSQLReader allows reading data for multiple symbols that are listed under the keyword \texttt{symbols}. 

and are fetched through the same connection. For each symbol the name must be given by the option name. The SQL query statement is passed through the option query. The symbol type can be specified using the option type. By default, every symbol is treated as a GAMS parameter. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent GAMSWriter. The GAMSWriter agent makes all read in symbols available outside the embedded Connect code.

Further, after solving the warehouse model, Connect can be used to export the results to tables in the SQL database.

```gams
Model swp 'simple warehouse problem' / all /

solve swp minimizing cost using lp;
```

```gams
EmbeddedCode Connect:
  - GAMSReader:
    readAll: True
  - Projection:
    name: stock.l(t)
    newName: stock_level(t)
  - Projection:
    name: sell.l(t)
    newName: sell_level(t)
  - Projection:
    name: buy.l(t)
    newName: buy_level(t)
  - PandasSQLWriter:
    connection: "sqlite:///whouse.db"
    symbols:
      - name: stock_level
        tableName: stock_level
        ifExists: replace
      - name: sell_level
        tableName: sell_level
        ifExists: replace
      - name: buy_level
        tableName: buy_level
        ifExists: replace
endEmbeddedCode
```

Here, we need to use execution-time embedded Connect code. The Connect agent GAMSReader imports all the variables into the Connect database. The PandasSQLWriter agent then writes each symbol to respective tables in the SQL database `whouse.db`. For example the stock level:

```
<table>
<thead>
<tr>
<th>t_0</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>q-1</td>
<td>100.0</td>
</tr>
<tr>
<td>q-2</td>
<td>10.0</td>
</tr>
<tr>
<td>q-3</td>
<td>10.0</td>
</tr>
<tr>
<td>q-4</td>
<td>10.0</td>
</tr>
</tbody>
</table>
```

The ifExists option allows to either append to an extending table or replace it with new data. By default, the value for ifExists is set to fails.
4.46 GAMS Connect

4.46.4.4 Complex Connect Example with Spreadsheets

The following example (inspired by the model \texttt{herves}) reads a 3-dimensional parameter from a spreadsheet that has one row index (\texttt{code}) at the left side of the table and the other row index \texttt{(labId)} at the right of the table. A column index \texttt{(cut)} is at the top of the table. The column index consists of floating-point numbers. The goal it to read the data into GAMS but modify the labels of some sets: Only the first two decimal digits of the elements in \texttt{cut} are significant. Moreover, the \texttt{labId} should be prefixed with an \texttt{L}. A new spreadsheet with the new labels should be written. The layout of the table should remain with the exception of moving the \texttt{labId} column also to the left. Here is a screenshot of the original table:

The following GAMS code uses a separate GAMS program (\texttt{getdata.gms}) to get the raw data from the original spreadsheet. Connect runs inside a compile-time embedded code section and uses the Connect agent \texttt{RawExcelReader} to get the raw Excel data. In some subsequent GAMS code the sets \texttt{rr} and \texttt{cut[Id]} as well as the parameter \texttt{raw} are filled knowing the layout of the table (the code is written in a way that the table can grow). This GAMS program gets executed and instructed to create a GDX file. In a compile-time embedded Connect section the relevant symbols (\texttt{rr, cut, and labdata}) are read from this GDX file. The \texttt{Projection} agent extracts the domain \texttt{labId} from the set \texttt{rr} and some Python code using Connect agent \texttt{PythonCode} makes the label adjustments and sorts the data nicely. The Python code uses the embedded Python methods \texttt{gams.set} to send the data to GAMS, so no GAMSWriter agent is necessary. In the main program at execution-time an embedded Connect code section exports the \texttt{labdata} parameter in the required form (after reading it from GAMS with the GAMSReader agent). Here is a screenshot of the resulting table:

In the remainder of the GAMS code another execution-time embedded Connect code is used to read the data back from the newly created spreadsheet using Connect agent \texttt{PandasExcelReader}. The set \texttt{rr} is created from parameter \texttt{labdata} using the \texttt{Projection} agent and everything is written back to GAMS with Connect agent \texttt{GAMSWriter}. The original data and the data from the newly created spreadsheet are exported to GDX (using \texttt{execute unload}) and compared to verify that the data is identical by calling \texttt{gdxdiff}. The entire code is listed below. This model is part of DataLib as model \texttt{connect01}.

```
Set code, labId, cut, rr(code<,labId);
parameter labdata(code,labid,cut);
$onEcho > getdata.gms
* Symbols for RawExcelReader
alias (u,*); Set s,w,r,c,ws(s,w),vs(s,r,c),vu(s,r,c,u); Parameter vf(s,r,c);
$onEmbeddedCode Connect:
 - RawExcelReader:
    file: labdata.xlsx
 - GAMSWriter:
    writeAll: True
$offEmbeddedCode
* Symbols to be filled
alias (*,code,labId,cut); Parameter raw(code,labid,cut); Set cutId, rr(code,labId)
Set cX(c,cut) 'column index', rX(r,code,labId) 'row index';
Singleton set cLast(c); Scalar lastPos;
loop(ws(s,‘ZAg’),
    lastPos = smax(vu(s,r,c,u), c.pos); cLast(c) = c.pos=lastPos;
    loop(r$(ord(r)>4),
        rX(r,code,labId) $= vu(s,r,’Cl’,code) and vu(s,r,cLast,labId));
    loop(c$(ord(c)>1 and not cLast(c)),
        cX(c,cut) $= vu(s,’R4′,c,cut));
    loop((rX(r,code,labId),cX(c,cut)),
        raw(code,labId,cut) = vf(s,r,c))
    loop(cX(c,cut),
        cutId(cut) = yes)
);
```
option rr(Intent); $offEcho
$call.checkErrorLevel gams getdata.gms lo=%gams.lo% gdx=getdata.gdx
$onEmbeddedCode
Connect:
- GDXReader:
  file: getdata.gdx
  symbols: [ {name: rr}, {name: raw}, {name: cutId, newName: cut} ]
- Projection:
  name: rr(code, labid)
  newName: labid(labid)
- PythonCode:
  code: |
    gams.set('labid', sorted(['L'+t[0] for t in connect.container.data['labid'].records.values], key=lambda t: int(t[1:])))
    gams.set('rr', sorted([(t[0], 'L'+t[1]) for t in connect.container.data['rr'].records.values], key=lambda t: int(t[0])))
    # Trim elements of set Cut to two decimal places
    gams.set('cut', sorted(['{:2f}'.format(float(t[0])) for t in connect.container.data['cut'].records.values], key=float))
    gams.set('labdata', [(t[0], 'L'+t[1], '{:.2f}'.format(float(t[2])), t[-1]) for t in connect.container.data['raw'].records.values])
$offEmbeddedCode
execute_unload 'labdata.gdx', labdata, cut, rr;

* Reintroduce 0 (zeros)
labdata(rr, cut) = labdata(rr, cut) + eps;
execute 'rm -f labdatanew.xlsx';
* Write new workbook with good table
EmbeddedCode
Connect:
- GAMSReader:
  symbols: [ {name: labdata} ]
- PandasExcelWriter:
  file: labdatanew.xlsx
  symbols:
    - name: labdata
      rowDimension: 2
      range: ZAg!A4
endEmbeddedCode
option clear=rr, clear=labdata;

EmbeddedCode
Connect:
- PandasExcelReader:
  file: labdatanew.xlsx
  symbols:
    - name: labdata
      rowDimension: 2
      columnDimension: 1
      range: ZAg!A4
- Projection:
  name: labdata(code, labid, cut)
  newName: rr(code, labid)
  asSet: True
- GAMSWriter:
  writeAll: True
endEmbeddedCode
execute_unload 'labdatanew.gdx', labdata, cut, rr;
execute.checkErrorLevel 'gdxdiff labdata.gdx labdatanew.gdx > %system.NullFile%';

4.46.4.5 Complex Connect Example with CSV Files

The following example shows how to read and write CSV files in Connect. It starts out with defining some data (stockprice) in a table statement in GAMS. With compile-time embedded Connect code utilizing the GAMSReader agent to bring this data into Connect and exporting it as a CSV file with agent CSVWriter. The GDXWriter agent also creates a GDX file with the data which is then used in a subsequent call to feed gdxdump that produces the same CSV file as CSVWriter. The text comparison tool diff is used to compare the two CSV files. The CSV file look as follows:

```
"date_0","AAPL","GOOG","MMM","MSFT","WMT"
"2012-20-11",12.124061,314.008026,60.966354,21.068886,46.991535
"2112-20-11",12.139372,311.741516,60.731037,20.850344,47.150307
"2212-20-11",12.203673,313.674286,61.467381,20.890808,46.991535
"2312-20-11",12.350039,315.387848,62.401108,21.068886,47.626663
"2712-20-11",12.448025,318.929565,62.461876,21.076981,47.499634
"2812-20-11",12.328911,318.656009,61.604042,20.899805,47.420238
"2912-20-11",12.404848,320.000549,62.332813,21.060795,47.626663
"3012-20-11",12.401172,321.744019,62.044331,21.012224,47.444057
```

In remainder of the example this CSV file is read back via the Connect agent CSVReader. The code also utilizes the tool csv2gdx to read the CSV file into a GDX file. The code compares the results of both methods. Csv2gdx also creates sets with the index elements as Dim1, Dim2, ... Therefore, Connect utilizes the Projection agent to extract the index sets date and symbol from the parameter stockprice as sets Dim1 and Dim2. The Connect agent GDXWriter creates a GDX file of the Connect database which then can be compared with the GDX file created by csv2gdx. The GDX comparison tool gdxdiff is used to compare the two GDX files. The entire code is listed below. This model is part of DataLib as model connect02.

```gams
Set date,symbol;
Table stockprice(date<,symbol<)
    AAPL  GOOG  MMM  MSFT  WMT
2012-20-11 12.124061  314.008026  60.966354  21.068886  46.991535
2112-20-11 12.139372  311.741516  60.731037  20.850344  47.150307
2212-20-11 12.203673  313.674286  61.467381  20.890808  46.991535
2312-20-11 12.350039  315.387848  62.401108  21.068886  47.626663
2712-20-11 12.448025  318.929565  62.461876  21.076981  47.499634
2812-20-11 12.328911  318.656009  61.604042  20.899805  47.420238
2912-20-11 12.404848  320.000549  62.332813  21.060795  47.626663
3012-20-11 12.401172  321.744019  62.044331  21.012224  47.444057;

* Use Connect CSVWriter to write GAMS data in CSV format moving the symbol index into the column (unstack: True)
$$\text{onEmbeddedCode Connect:}$$
  - GAMSReader:
    symbols: [ {name: stockprice} ]
  - GDXWriter:
    file: sp_connect.gdx
    writeAll: True
  - CSVWriter:
    file: sp_connect.csv
    name: stockprice
```
* Use gdxdump to create a CSV file and text compare the Connect and gdxdump CSV files
$call.checkErrorLevel gdxdump sp_connect.gdx output=sp_gdxdump.csv symb=stockprice format=csv columnDimension=Y > %system.NullFile%
$call.checkErrorLevel diff -q sp_connect.csv sp_gdxdump.csv > %system.nullFile%

* Use Connect CSVReader to read the newly created CSV file and deposit the result in a csv2gdx compatible format
$onEmbeddedCode Connect:
- CSVReader:
  file: sp_connect.csv
  name: stockprice
  indexColumns: 1
  valueColumns: "2:lastCol"
- Projection:
  name: stockprice(date,symbol)
  newName: Dim1(date)
  asSet: True
- Projection:
  name: stockprice(date,symbol)
  newName: Dim2(symbol)
  asSet: True
- GDXWriter:
  file: sp_connect.gdx
  writeAll: True
$offEmbeddedCode

* Use csv2gdx to create a GDX file and compare the Connect and csv2gdx GDX files
$call.checkErrorLevel csv2gdx sp_connect.csv output=sp_csv2gdx.gdx id=stockprice index=1 value=2..lastCol useHeader=y > %system.NullFile%
$call.checkErrorLevel gdxdiff sp_connect.gdx sp_csv2gdx.gdx > %system.NullFile%

4.46.5 Connect Agents

4.46.5.1 CSVReader

The CSVReader allows reading a symbol from a specified CSV file into the Connect database. Its implementation is based on the pandas.DataFrame class and its I/O API method read_csv.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoColumn</td>
<td>None</td>
<td>Generate automatic column names.</td>
</tr>
<tr>
<td>autoRow</td>
<td>None</td>
<td>Generate automatic row labels.</td>
</tr>
<tr>
<td>decimalSeparator</td>
<td>(period)</td>
<td>Specify a decimal separator.</td>
</tr>
<tr>
<td>fieldSeparator</td>
<td>(comma)</td>
<td>Specify a field separator.</td>
</tr>
<tr>
<td>file</td>
<td>None</td>
<td>Specify a CSV file path.</td>
</tr>
<tr>
<td>header</td>
<td>inferred</td>
<td>Indicate if the first row in the CSV file should be used as the column names in the DataFrame.</td>
</tr>
<tr>
<td>indexColumns</td>
<td>None</td>
<td>Specify columns to use as the row labels.</td>
</tr>
<tr>
<td>indexSubstitutions</td>
<td>None</td>
<td>Dictionary used for substitutions in the index columns.</td>
</tr>
<tr>
<td>name</td>
<td>None</td>
<td>Specify a symbol name for the Connect database.</td>
</tr>
<tr>
<td>names</td>
<td>None</td>
<td>List of column names to use.</td>
</tr>
<tr>
<td>quoting</td>
<td>0</td>
<td>Control field quoting behavior.</td>
</tr>
</tbody>
</table>
**Option** | **Default** | **Description**
---|---|---
readCSVArguments | None | Dictionary containing keyword arguments for the `pandas.read_csv` method.
skipRows | None | Specify the rows to skip or the number of rows to skip.
stack | inferred | Stacks the column names to index.
textColumns | None | Specify columns to get the set element text from.
textSubstitutions | None | Dictionary used for substitutions in the text columns.
thousandsSeparator | None | Specify a thousands separator.
trace | inherited | Specify the trace level for debugging output.
valueColumns | None | Specify columns to get the values from.
valueSubstitutions | None | Dictionary used for substitutions in the value columns.

Detailed description of the options:

**autoColumn = string** (optional)

Generate automatic column names. The `autoColumn` string is used as the prefix for the column label numbers. This option overrides the use of a `header` or `names`. However, if there is a header row, one must skip the row by enabling `header` or using `skipRows`.

**autoRow = string** (optional)

Generate automatic row labels. The `autoRow` string is used as the prefix for the row label numbers. The generated unique elements will be used in the first index position shifting other elements to the right. Using `autoRow` can be helpful when there are no labels that can be used as unique elements but also to store entries that would be a duplicate entry without a unique row label.

**decimalSeparator = string** (default=`.`)

Specify a decimal separator. [. (period), , (comma)]

**fieldSeparator = string** (default=`,`)

Specify a field separator. [, (comma), ; (SemiColon), \t (Tab)]

**file = string** (required)

Specify a CSV file path.

**header = boolean** (optional)

Indicate if the first row in the CSV file should be used as the column names in the `DataFrame`. Default behavior is to infer the column names: if no `names` are passed the behavior is identical to `header=True` and column names are inferred from the first line of data, if column names are passed explicitly then the behavior is identical to `header=False`. Explicitly pass `header=True` to be able to replace existing names. Note that missing column names are filled with `Unnamed: n` (where `n` is the nth column (zero based) in the DataFrame). Hence, reading the CSV file:

```
,i1,1,2
,i2,3,4
,j1,5,6
```

results in the following 2-dimensional parameter:
indexColumns = list or string (optional)

Specify columns to use as the row labels. The columns can either be given as column positions or column names. Column positions can be represented as an integer, a list of integers or a string. For example: indexColumns: [1, 2, 3, 4, 6] or indexColumns: "1:4, 6". The symbolic constant lastCol can be used with the string representation: "2:lastCol". Note that the usage of lastCol requires header: True or names given. Column names can be represented as a list of strings. For example: indexColumns: ["i1","i2"]. Note that indexColumns and valueColumns/textColumns and must either be given as positions or names not both.

By default the pandas.read_csv method interprets the following indices as NaN: "", ".N/A", ".N/A", ".-1.#IND", ".-1.#QNAN", ".-NaN", ".-nan", ".1.#IND", ".1.#QNAN", ".<NA>", ".N/A", ".NA", ".NULL", ".NaN", ".n/a", ".nan", ".null". The default can be changed by specifying pandas.read_csv arguments keep_default_na and na_value via readCSVArguments. Rows with indices that are interpreted as NaN will be dropped automatically. The indexSubstitutions option allows to remap NaN entries in the index columns.

indexSubstitutions = dictionary (optional)

Dictionary used for substitutions in the index columns. Each key in indexSubstitutions is replaced by its corresponding value. This option allows arbitrary replacements in the index columns of the DataFrame including stacked column names. Consider the following CSV file:

```
i1,j1,2.5
i1,,1.7
i2,j1,1.8
i2,,1.4
```

Reading this data into a 2-dimensional parameter results in a parameter with NaN entries dropped:

```
j1
i1  2.500
i2  1.800
```

By specifying indexSubstitutions: { .nan: j2 } we can substitute NaN entries by j2:

```
j1         j2
i1  2.500   1.700
i2  1.800   1.400
```

name = string (required)

Specify a symbol name for the Connect database. Note that each symbol in the Connect database must have a unique name.

names = list (optional)

List of column names to use. If the file contains a header row, then you should explicitly pass header=True to override the column names. Duplicates in this list are not allowed.
quoting = integer (default=0)

Control field quoting behavior. Use QUOTE_MINIMAL (0), QUOTE_ALL (1),
QUOTE_NONNUMERIC (2) or QUOTE_NONE (3). QUOTE_NONNUMERIC (2) in-
structs the reader to convert all non-quoted fields to type float. QUOTE_NONE (3) instructs
reader to perform no special processing of quote characters.

readCSVArguments = dictionary (optional)

Dictionary containing keyword arguments for the pandas.read_csv method. Not all argu-
ments of that method are exposed through the YAML interface of the CSVReader agent. By
specifying toCSVArguments, it is possible to pass arguments directly to the pandas.read_csv
method that is used by the CSVReader agent.

skipRows = list or integer (optional)

Specify the rows to skip (list) or the number of rows to skip (integer). For example: skipRows:
[1, 3] or skipRows: 5.

stack = boolean (optional)

Stacks the column names to index. Defaults to True if there is more than one value/text
column, otherwise False. Note that missing column names are filled withUnnamed: n (where
n is the nth column (zero based) in the DataFrame).

textColumns = list or string (optional)

Specify columns to get the set element text from. The columns can be given as column
positions or column names. Column positions can be represented as a integer, a list of
integers or a string. For example: textColumns: 1, textColumns: [1, 2, 3, 4, 6] or
textColumns: "1:4, 6". The symbolic constant lastCol can be used with the string
representation: "2:lastCol". Note that the usage of lastCol requires header: True or
names given. Column names can be represented as a list of strings. For example: textColumns:
["i1", "i2"]. Note that textColumns and indexColumns must either be given as positions
or names not both.

By default the pandas.read_csv method interprets the following text as NaN: ", ", ",N/A",
"#N/A", "#NA", ""-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN",
"<NA>"", "N/A", "NA", "NULL", "NaN", "n/a", "nan", "null". The default can be
changed by specifying pandas.read_csv arguments keep_default_na and na_value via
readCSVArguments. Rows with texts that are interpreted as NaN will be dropped auto-
matically. The textSubstitutions option allows to remap NaN entries in the text columns.

textSubstitutions = dictionary (optional)

Dictionary used for substitutions in the text columns. Each key in textSubstitutions is
replaced by its corresponding value. While it is possible to make arbitrary replacements
this is especially useful for controlling sparse/dense reading. The default reading behavior is
sparse since rows with text that is interpreted as NaN are dropped automatically. Consider the
following CSV file:

i1,text1
i2,
i3,text3

thousandsSeparator = string (optional)

Specify a thousands separator.
trace = integer (optional)

Specify the trace level for debugging output. For \texttt{trace} > 1 some scalar debugging output will be written to the log. For \texttt{trace} > 2 the intermediate data frames will be written abbreviated to the log. For \texttt{trace} > 3 the intermediate data frames will be written entirely to the log (potentially large output). If \texttt{trace} has not been set, the \texttt{trace} value, set by the \texttt{Options} agent, will be used.

valueColumns = list or string (optional)

Specify columns to get the values from. The columns can be given as column positions or column names. Column positions can be represented as an integer, a list of integers or a string. For example: \texttt{valueColumns: 1, valueColumns: [1, 2, 3, 4, 6] or valueColumns: "1:4, 6"}. The symbolic constant \texttt{lastCol} can be used with the string representation: "2:lastCol". Note that the usage of \texttt{lastCol} requires \texttt{header: True} or \texttt{names} given. Column names can be represented as a list of strings. For example: \texttt{valueColumns: ["i1","i2"]}. Note that \texttt{valueColumns} and \texttt{indexColumns} must either be given as positions or names not both.

By default the \texttt{pandas.read.csv} method interprets the following values as NaN: ",", ",#N/A", ",#N/A N/A", ",#NA", ",-1.#IND", ",-1.#QNAN", ",-NaN", ",-nan", ",1.#IND", ",1.#QNAN", ",<NA>\", ",N/A\", ",NA\", ",NULL\", ",NaN\", ",n/a\", ",nan\", ",null\". The default can be changed by specifying \texttt{pandas.read.csv} arguments \texttt{keep_default_na} and \texttt{na_value} via \texttt{readCSVArguments}. Rows with values that are interpreted as NaN will be dropped automatically. Changing the default of values that are interpreted as NaN is useful if, e.g., "NA" values should not be dropped but interpreted as GAMS special value NA. Moreover, the \texttt{valueSubstitutions} option allows to remap NaN entries in the value columns.

valueSubstitutions = dictionary (optional)

Dictionary used for substitutions in the value columns. Each key in \texttt{valueSubstitutions} is replaced by its corresponding value. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading. All NaN entries are removed automatically by default which results in a sparse reading behavior. Consider the following CSV file:

\begin{verbatim}
  i1,j1,
i1,j2,1.7
i2,j1,
i2,j2,1.4
\end{verbatim}

Reading this data into a 2-dimensional parameter results in a sparse parameter with all NaN entries removed:

\begin{verbatim}
  j2
  i1 1.700
  i2 1.400
\end{verbatim}

By specifying \texttt{valueSubstitutions: { .nan: eps } we get a dense representation where all NaN entries are replaced by GAMS special value EPS:

\begin{verbatim}
  j1   j2
  i1   EPS 1.700
  i2   EPS 1.400
\end{verbatim}
Beside $\texttt{eps}$ there are the following other GAMS special values that can be used by specifying their string representation: $\texttt{inf}$, $-\texttt{inf}$, $\texttt{eps}$, $\texttt{na}$, and $\texttt{undef}$. See the GAMS Transfer documentation for more information.

Reading this data into a 1-dimensional set results in a sparse set in which all NaN entries (those that do not have any set element text) are removed:

'\texttt{i1}' 'text 1',
'\texttt{i3}' 'text 3'

By specifying textSubstitutions: \{ .nan: '' \} we get a dense representation:

'\texttt{i1}' 'text 1',
'\texttt{i2}',
'\texttt{i3}' 'text 3'

It is also possible to use textSubstitutions in order to interpret the set element text. Let's assume we have the following CSV file:

,.j1,j2,j3
i1,Y,Y,Y
i2,Y,Y,N
i3,0,Y,Y

Reading this data into a 2-dimensional set results in a dense set:

'\texttt{i1}'.\'j1' Y,
'\texttt{i1}'.\'j2' Y,
'\texttt{i1}'.\'j3' Y,
'\texttt{i2}'.\'j1' Y,
'\texttt{i2}'.\'j2' Y,
'\texttt{i2}'.\'j3' N,
'\texttt{i3}'.\'j1' 0,
'\texttt{i3}'.\'j2' Y,
'\texttt{i3}'.\'j3' Y

By specifying textSubstitutions: \{ 'N': .nan, '0': .nan \} we replace all occurrences of N and 0 by NaN which gets dropped automatically:

'\texttt{i1}'.\'j1' Y,
'\texttt{i1}'.\'j2' Y,
'\texttt{i1}'.\'j3' Y,
'\texttt{i2}'.\'j1' Y,
'\texttt{i2}'.\'j2' Y,
'\texttt{i3}'.\'j2' Y,
'\texttt{i3}'.\'j3' Y

### 4.46.5.2 CSVWriter

The CSVWriter allows writing a symbol in the Connect database to a specified CSV file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimalSeparator</td>
<td>. (period)</td>
<td>Specify a decimal separator.</td>
</tr>
<tr>
<td>file</td>
<td>None</td>
<td>Specify a CSV file path.</td>
</tr>
<tr>
<td>fieldSeparator</td>
<td>, (comma)</td>
<td>Specify a field separator.</td>
</tr>
<tr>
<td>header</td>
<td>True</td>
<td>Indicate if the header will be written.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>quoting</td>
<td>0</td>
<td>Control field quoting behavior.</td>
</tr>
<tr>
<td>setHeader</td>
<td>None</td>
<td>Specify a string that will be used as the header.</td>
</tr>
<tr>
<td>skipElementText</td>
<td>False</td>
<td>Indicate if the set element text will be skipped.</td>
</tr>
<tr>
<td>toCSVArguments</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the pandas.to_csv method.</td>
</tr>
<tr>
<td>trace</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>unstack</td>
<td>False</td>
<td>Indicate if the last dimension will be used as a header row.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**decimalSeparator = string (default=.)**

Specify a decimal separator. [ . (period), , (comma)]

**file = string (required)**

Specify a CSV file path.

**fieldSeparator = string (default=,)**

Specify a field separator. [ , (comma), ; (SemiColon), \t (Tab)]

**header = boolean (default=True)**

Indicate if the header will be written.

**name = string (required)**

Specify the name of the symbol in the Connect database.

**quoting = integer (default=0)**

Control field quoting behavior. Use QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) or QUOTE_NONE (3). QUOTE_MINIMAL (0) instructs the writer to only quote those fields which contain special characters such as fieldSeparator. QUOTE_ALL (1) instructs the writer to quote all fields. QUOTE_NONNUMERIC (2) instructs the writer to quote all non-numeric fields. QUOTE_NONE (3) instructs the writer to never quote fields.

**setHeader = string (optional)**

Specify a string that will be used as the header. If an empty header is desired, the string can be empty.

**skipElementText = boolean (default=False)**

Indicate if the set element text will be skipped. If False, the set element text will be written in the last column of the CSV file.

**toCSVArguments = dictionary (optional)**

Dictionary containing keyword arguments for the pandas.to_csv method. Not all arguments of that method are exposed through the YAML interface of the CSVWriter agent. By specifying toCSVArguments, it is possible to pass arguments directly to the pandas.to_csv method that is used by the CSVWriter agent.

**trace = integer (optional)**

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

**unstack = boolean (default=False)**

Indicate if the last dimension will be used as a header row.
4.46.5.3 DomainWriter

The DomainWriter agent allows to rewrite domain information for existing Connect symbols and helps dealing with domain violations.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropDomainViolations</td>
<td>global/symbol</td>
<td>False</td>
<td>Indicate how to deal with domain violations.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>writeAll</td>
<td>global</td>
<td>False</td>
<td>Indicate if all symbols in the Connect database will be treated with the global dropDomainViolations setting.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**dropDomainViolations** = boolean (global)/string (symbol) (default=False)

The Connect symbols might have some domain violations. This agent allows to drop these domain violations so a write to GAMS or GDX works properly. Setting the global dropDomainViolations: True together with writeAll: True will drop all domain violations from all Connect symbols. The symbols section allows to also drop domain violations. In the symbols section the dropDomainViolations attribute need a string value. Possible string values are before, after, "True", and "False". If the attribute has not been set for the symbol, the attribute is inherited from the global attribute. The value before means that domain violations are dropped before a new domain is applied, see attribute name. The value after means that domain violations are dropped after a new domain is applied. The value "True" means that domain violations are dropped before and after a new domain is applied. "False" means to not drop domain violations.

**name** = string (required)

Specify a symbol name with index space for the Connect database. name requires the format symname(i1,i2,...,iN). The list of indices needs to coincide with the names of the actual GAMS domain sets for a regular domain. A relaxed domain is set if the index is quoted. For example name: x(i,’j’) means that for the first index a regular domain with domain set i is established, while for the second index the universal domain * is used and a relaxed domain name j is set.

**trace** = integer (optional)

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

**writeAll** = boolean (default=False)

Indicate if all symbols in the Connect database will be treated according to global attribute dropDomainViolations.

4.46.5.4 GAMSReader

The GAMSReader allows reading symbols from the GAMS database into the Connect database. Without GAMS context (e.g. when running the gamsconnect script from the command line) this agent is not available and its execution will result in an exception.
<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the GAMS database.</td>
</tr>
<tr>
<td>newName</td>
<td>symbol</td>
<td>None</td>
<td>Specify a new name for the symbol in the Connect database.</td>
</tr>
<tr>
<td>readAll</td>
<td>global</td>
<td>False</td>
<td>Indicate if all symbols in the GAMS database will be read into the Connect database.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**name = string (required)**

Specify the name of the symbol in the GAMS database.

**newName = string (optional)**

Specify a new name for the symbol in the Connect database. Each symbol in the Connect database must have a unique name.

**readAll = boolean (default=True)**

Indicate if all symbols in the GAMS database will be read into the Connect database. If True, read all symbols into the Connect database and ignore symbol options.

**trace = integer (optional)**

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

### 4.46.5.5 GAMSWriter

The GAMSWriter allows writing symbols in the Connect database to the GAMS database. Without GAMS context (e.g. when running the `gamsconnect` script from the command line) and as part of the `connectOut` command line option this agent is not available and its execution will result in an exception.
Detailed description of the options:

\( \text{domainCheckType} = \text{string} \) (default=\text{default})

Specify if Domain Checking is applied (\text{checked}) or if records that would cause a domain violation are filtered (\text{filtered}). If left at \text{default} it depends on the setting of \$on/offFiltered if GAMS does a filtered load or checks the domains during compile time. During execution time \text{default} is the same as \text{filtered}.

\( \text{duplicateRecords} = \text{string} \) (default=\text{all})

The Connect container can hold multiple records even for the same indexes. This is only a problem when exchanging the data with GAMS (and GDX). The attribute determines how the agent deals with duplicate records. With the default of \text{all} the GAMSWriter will fail in case duplicate records exist. With \text{first} the first record will be written to GAMS, with \text{last} the last record written to GAMS. With \text{none} none of the duplicate records will be written to GAMS. Note that the agent currently deals with duplicate records in a case sensitive way.

\( \text{mergeType} = \text{string} \) (default=\text{default})

Specify if data in a GAMS symbol is merged (\text{merge}) or replaced (\text{replace}). If left at \text{default} it depends on the setting of \$on/offMulti\{R\} if GAMS does a merge, replace, or triggers an error during compile time. During execution time \text{default} is the same as \text{merge}.

\( \text{name} = \text{string} \) (required)

Specify the name of the symbol in the Connect database.

\( \text{newName} = \text{string} \) (optional)

Specify a new name for the symbol in the GAMS database. Note, each symbol in the GAMS database must have a unique name.

\( \text{trace} = \text{integer} \) (optional)

Specify the trace level for debugging output. For \text{trace} \geq 1 some scalar debugging output will be written to the log. For \text{trace} > 2 the intermediate data frames will be written abbreviated to the log. For \text{trace} > 3 the intermediate data frames will be written entirely to the log (potentially large output). If \text{trace} has not been set, the \text{trace} value, set by the Options agent, will be used.

\( \text{writeAll} = \text{boolean} \) (default=\text{False})

Indicate if all symbols in the Connect database will be written to the GAMS database. If \text{True}, write all symbols to the GAMS database and ignore symbol options.

### 4.46.5.6 GDXReader

The GDXReader allows reading symbols from a specified GDX file into the Connect database.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>global</td>
<td>None</td>
<td>Specify a GDX file path.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the GDX file.</td>
</tr>
<tr>
<td>newName</td>
<td>symbol</td>
<td>None</td>
<td>Specify a new name for the symbol in the Connect database.</td>
</tr>
<tr>
<td>readAll</td>
<td>global</td>
<td>\text{False}</td>
<td>Indicate if all symbols in the GDX file will be read into the Connect database.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
</tbody>
</table>
Detailed description of the options:

**file = string (required)**

Specify a GDX file path.

**name = string (required)**

Specify the name of the symbol in the GDX file.

**newName = string (optional)**

Specify a new name for the symbol in the Connect database. Each symbol in the Connect database must have a unique name.

**readAll = boolean (default=False)**

Indicate if all symbols in the GDX file will be read into the Connect database. If True, read all symbols into the Connect database and ignore symbol options.

**trace = integer (optional)**

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

### 4.46.5.7 GDXWriter

The GDXWriter allows writing symbols in the Connect database to a specified GDX file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicateRecords</td>
<td>global/symbol</td>
<td>all</td>
<td>Specify how to deal with duplicate records.</td>
</tr>
<tr>
<td>file</td>
<td>global</td>
<td>None</td>
<td>Specify a GDX file path.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>newName</td>
<td>symbol</td>
<td>None</td>
<td>Specify a new name for the symbol in the GDX file.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>writeAll</td>
<td>global</td>
<td>False</td>
<td>Indicate if all symbols in the Connect database will be written to the GDX file.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**duplicateRecords = string (default=all)**

The Connect container can hold multiple records even for the same indexes. This is only a problem when exchanging the data with GDX (and GAMS). The attribute determines how the agent deals with duplicate records. With the default of all the GDXWriter will fail in case duplicate records exist. With **first** the first record will be written to GDX, with **last** the last record written to GDX. With **none** none of the duplicate records will be written to GDX. Note that the agent currently deals with duplicate records in a case sensitive way.

**file = string (required)**
Specify a GDX file path.

**name = string (required)**

Specify the name of the symbol in the Connect database.

**newName = string (optional)**

Specify a new name for the symbol in the GDX file. Note, each symbol in the GDX file must have a unique name.

**trace = integer (optional)**

Specify the trace level for debugging output. For \( \text{trace} > 1 \) some scalar debugging output will be written to the log. For \( \text{trace} > 2 \) the intermediate data frames will be written abbreviated to the log. For \( \text{trace} > 3 \) the intermediate data frames will be written entirely to the log (potentially large output). If \( \text{trace} \) has not been set, the \( \text{trace} \) value, set by the Options agent, will be used.

**writeAll = boolean (default=False)**

Indicate if all symbols in the Connect database will be written to the GDX file. If \text{True}, write all symbols to the GDX file and ignore symbol options.

### 4.46.5.8 Options

The Options agent allows to set more general options that can affect the Connect database and other Connect agents. More specifically, the value of an option set via the Options agent can be inherited as a default value to Connect agents that utilize the considered option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caseSensitiveLookup</td>
<td>True</td>
<td>Specify how to lookup symbols specified by attribute name and others in the Connect database.</td>
</tr>
<tr>
<td>debug</td>
<td>0</td>
<td>If set to &gt;0 a full trace is written to stdout in case of an exception.</td>
</tr>
<tr>
<td>trace</td>
<td>0</td>
<td>Specify the trace level for debugging output.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**caseSensitiveLookup = boolean (default=True)**

Specify how to lookup symbols in the Connect database. Unlike GAMS, Connect uses case sensitive symbol names hence one needs to pay attention to symbol name casing when the symbol name is specified in an agent description, e.g. via the \text{name} attribute. If the caseSensitiveLookup attribute is set \text{False} Connect will also find symbols if their names match in an case-insensitive way. If multiple symbols with same lower case name exist in the Connect database, the symbol that matches exactly is found. If none of the symbols match exactly, then one of symbols is found but it is unclear which one. For example, symbols ABC and abc exist in the Connect database. If \text{name}: ABC is given symbol ABC will be used, but if \text{name}: aBc is given either abc or ABC is returned. Multiple symbols with the same lower-case name should be avoided, they cause problems e.g. when writing to GAMS or GDX with attribute writeAll: \text{True}.

**debug = integer (default=0)**
A Connect error triggers an exception. By default only the exception message is provided to the user. In case the `debug` attribute is set to an integer larger than 0, a full trace with the calling stack is printed to stdout.

\[
\text{trace} = \text{integer} \ (\text{default}=0)
\]

Specify the trace level for debugging output. A trace level of 0 (default) means no debugging output. For `trace > 0` the Connect database will write some scalar debugging output to the log. The debugging output of Connect agents depends on their implementation of `trace`, please refer to the corresponding documentation.

### 4.46.5.9 PandasExcelReader

The PandasExcelReader agent allows to read symbols from an Excel file into the Connect database. Its implementation is based on the `pandas.DataFrame` class and its I/O API method `read_excel`. The PandasExcelReader primarily aims to read spreadsheets that have been written by its counterpart - the PandasExcelWriter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>columnDimension</code></td>
<td>global/symbol</td>
<td>1</td>
<td>Column dimension of the symbol.</td>
</tr>
<tr>
<td><code>drop</code></td>
<td>symbol</td>
<td>None</td>
<td>Specify a string for dropping each row containing it in one of its labels.</td>
</tr>
<tr>
<td><code>excelFileArguments</code></td>
<td>global</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the <code>pandas.ExcelFile</code> constructor.</td>
</tr>
<tr>
<td><code>file</code></td>
<td>global</td>
<td>None</td>
<td>Specify an Excel file path.</td>
</tr>
<tr>
<td><code>multiColumnBlankRow</code></td>
<td>global/symbol</td>
<td>True</td>
<td>Indicator for existence of blank row after the column indexes (for <code>columnDimension&gt;1</code>).</td>
</tr>
<tr>
<td><code>name</code></td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td><code>range</code></td>
<td>symbol</td>
<td>None</td>
<td>Specify the Excel range of a symbol.</td>
</tr>
<tr>
<td><code>readExcelArguments</code></td>
<td>symbol</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the <code>pandas.read_excel</code> method.</td>
</tr>
<tr>
<td><code>rowDimension</code></td>
<td>global/symbol</td>
<td>1</td>
<td>Row dimension of the symbol.</td>
</tr>
<tr>
<td><code>trace</code></td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td><code>type</code></td>
<td>global/symbol</td>
<td>par</td>
<td>Control the symbol type.</td>
</tr>
<tr>
<td><code>valueSubstitutions</code></td>
<td>symbol</td>
<td>None</td>
<td>Dictionary used for mapping in the <code>value</code> column of the <code>DataFrame</code>.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**columnDimension = integer (default=1)**

Column dimension: the number of rows in the data range that will be used to define the labels for columns. The first `columnDimension` rows of the data range will be used for labels.

**drop = string (optional)**

Specify a string for dropping each row containing it in one of its labels. The specified string is interpreted as a regular expression.

**excelFileArguments = dictionary (optional)**

Dictionary containing keyword arguments for the `pandas.ExcelFile` constructor.
4.46 GAMS Connect

file = string (required)

Specify an Excel file path.

multiColumnBlankRow = boolean (default=True)

For symbols where more than one dimension is in the columns, i.e. \( \text{columnDimension} > 1 \) the PandasExcelReader expects a blank row before the data starts. This is also the shape the PandasExcelWriter writes:

If multiColumnBlankRow is set to False, the PandasExcelReader expects for table with \( \text{columnDimension} > 1 \) that this blank line is missing. This works properly with the exception of the following corner case:

together with the following Connect instructions

- PandasExcelReader:
  - file: myfile.xlsx
  - symbols:
    - name: s
      rowDimension: 1
      columnDimension: 2
      range: B2:E6
      type: set
      multiColumnBlankRow: False
      valueSubstitutions: { .nan: '' } # read dense

one would expect that the Connect database contains a set with the following elements \( k_1^*k_3.(i_1.j_1,i_2.j_2,i_3.j_3) \) but Pandas interprets the row 4 (because it is entirely blank) as the row with the index name. Hence the data starts in row 5 and the Connect database is missing the \( k_1 \) records: \( k_2^*k_3.(i_1.j_1,i_2.j_2,i_3.j_3) \). In such a case one either needs the blank row between column indexes and data or manages to have this row not entirely empty (which can even be done outside the specified Excel range).

name = string (required)

Specify a symbol name for the Connect database. Note that each symbol in the Connect database must have a unique name.

range = string (required)

Specify the Excel range of a symbol using the format sheet!range. range can be either a single cell also known as open range (north-west corner like B2) or a full range (north-west and south-east corner like B2:D4). For symbols with \( \text{columnDimension}=0 \) and/or \( \text{rowDimension}=0 \), the ending row and/or ending column or the open range can be deduced and is used to restrict the data area.

readExcelArguments = dictionary (optional)

Dictionary containing keyword arguments for the pandas.read_excel method. Not all arguments of that method are exposed through the YAML interface of the PandasExcelReader agent. By specifying readExcelArguments, it is possible to pass arguments directly to the pandas.read_excel method that is used by the PandasExcelReader agent.

rowDimension = integer (default=1)

Row dimension: the number of columns in the data range that will be used to define the labels for the rows. The first \( \text{rowDimension} \) columns of the data range will be used for the labels.
trace = integer (optional)

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

type = string (default=par)

Control the symbol type. Supported symbol types are par for parameters and set for sets.

valueSubstitutions = dictionary (optional)

Dictionary used for mapping in the value column of the DataFrame. Each key in valueSubstitutions is replaced by its corresponding value. The replacement is only performed on the value column of the DataFrame which is the numerical value in case of a GAMS parameter and the set element text in case of a GAMS set. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading. All NaN entries are removed automatically by default which results in a sparse reading behavior. Let's assume we have the following spreadsheet:

Reading this data into a 2-dimensional parameter results in a sparse parameter in which all NaN entries are removed:

'\text{i1}'.'\text{j1}' 2.5,
'\text{i1}'.'\text{j2}' 1.7,
'\text{i2}'.'\text{j2}' 1.8,
'\text{i2}'.'\text{j3}' 1.4

By specifying valueSubstitutions: \{ .\text{nan}: \text{eps} \} we get a dense representation in which all NaN entries are replaced by GAMS special value EPS:

'\text{i1}'.'\text{j1}' 2.5,
'\text{i1}'.'\text{j2}' 1.7,
'\text{i1}'.'\text{j3}' Eps,
'\text{i2}'.'\text{j1}' Eps,
'\text{i2}'.'\text{j2}' 1.8,
'\text{i2}'.'\text{j3}' 1.4

Beside eps there are the following other GAMS special values that can be used by specifying their string representation: inf, -inf, eps, na, and undef. See the GAMS Transfer documentation for more information.

Let's assume we have data representing a GAMS set:

Reading this data into a 1-dimensional set results in a sparse set in which all NaN entries (those that do not have any set element text) are removed:

'\text{i1}' 'text 1',
'\text{i3}' 'text 3'

By specifying valueSubstitutions: \{ .\text{nan}: '' \} we get a dense representation:

'\text{i1}' 'text 1',
'\text{i2}',
'\text{i3}' 'text 3'
It is also possible to use `valueSubstitutions` in order to interpret the set element text. Let’s assume we have the following Excel data:

Reading this data into a 2-dimensional set results in a dense set:

```
'i1'.'j1' No,
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i2'.'j3' Y,
'i3'.'j1' Y,
'i3'.'j2' Y,
'i3'.'j3' N
```

By specifying `valueSubstitutions: { 'N': .nan, 'No': .nan }` we replace all occurrences of `N` and `No` by `nan` which gets dropped automatically. Note that `No` has to be quotes in order not to be interpreted as `False` by the YAML parser:

```
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i2'.'j3' Y,
'i3'.'j1' Y,
'i3'.'j2' Y
```

### 4.46.5.9.1 Fundamentals of Reading Data with PandasExcelReader

As mentioned at the start of this section, the PandasExcelReader works best with tables written by the PandasExcelWriter with a full range (north-west and south-east corner) specification. Nevertheless, the PandasExcelReader can also process tables not precisely in the format and shape given by PandasExcelWriter and also works with an open range (north-west corner only) specification. While the PandasExcelReader shares some functionality with the tool `gdxxrw` there are also significant differences and this section explains some of the perhaps unexpected behavior of this Connect agent.

- Symbols with `rowDimension=0` and/or `columnDimension=0` have an artificial index in 0-dim index and the range specification need to include this artificial index. For example, if one wants to read a scalar, there are two artificial indexes `rval` and `cval` and the north-west corner starts in the artificial index row and column, i.e. range is `B2` or `B2:C3` in the following example:

  The names of the artificial indexes are irrelevant, they can even be blank. But as a consequence of the artificial indexes, PandasExcelReader cannot read a scalar that is located in row 1 or column A. Similarly, symbols with `dim>0` but either `rowDimension` or `columnDimension` equal to 0 need an artificial index:

  Again, the name of the artificial index is irrelevant, it can even be blank. In both examples, the range starts in north-west corner `B2`. For symbols with `rowDimension>0` and `columnDimension>0` there is no artificial index.

- Tables with more than one index in the columns can be read best if the column headers and the data are separated by a blank line (this is the way PandasExcelWriter writes such tables). The PandasExcelReader attribute `multiColumnBlankRow` allows some control. See the option descriptions for details.

- Blank data is read as `nan` (not a number) and such records are dropped before the data is written to the Connect database. The `valueSubstitutions` attribute of PandasExcelReader allows to remap `nan` to other values. Blank index positions are treated very differently. The behavior depends on multiple factors.
If PandasExcelReader encounters a blank index at the beginning of the rows it fills the index with `nan`. With a blank row index in the middle of the table (i.e. if there was a good label before in this column), PandasExcelReader repeats the previous index of this column. In case of `columnDimension=1` blank column indexes are filled with `Unnamed: n` (where n in the nth column (zero based) in the sheet). Hence the following table is transformed into the following pandas.DataFrame (use PandasExcelReader attribute `trace>2` to print intermediate data frames):

<table>
<thead>
<tr>
<th>j1</th>
<th>j3</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i2</td>
<td>i2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>6.0</td>
</tr>
</tbody>
</table>

which arrives after dropping the `nan` values and indexes in the Connect database (here a display as a GAMS parameter) as:

<table>
<thead>
<tr>
<th>j1</th>
<th>j3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2.i2</td>
<td>2</td>
</tr>
<tr>
<td>i2.i3</td>
<td>3</td>
</tr>
<tr>
<td>i5.i3</td>
<td>5</td>
</tr>
<tr>
<td>i6.i6</td>
<td>6</td>
</tr>
</tbody>
</table>

In case of `columnDimension>1` blank column index at the beginning of the column index are filled with `Unnamed: n_level_k` (where n in the nth column (zero based) in the sheet and k is the column dimension). With a blank column index in the middle of the index columns (i.e. if there was a good label before in this row), PandasExcelReader repeats the previous index but might add a suffix `.1`, `.2`, ... to disambiguate the column names. Hence the following table is transformed into the following pandas.DataFrame (use PandasExcelReader attribute `trace>2` to print intermediate data frames):

<table>
<thead>
<tr>
<th>NaN</th>
<th>NaN</th>
<th>NaN</th>
<th>NaN</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2</td>
<td>i2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

which arrives after dropping the `nan` values and indexes in the Connect database (here a display as a GAMS parameter) as:

<table>
<thead>
<tr>
<th>NaN</th>
<th>NaN</th>
<th>NaN</th>
<th>NaN</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2.i2</td>
<td>i2.i3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The PandasExcelReader attribute `drop` helps to get rid of unwanted indexes (e.g. `drop: Unnamed` or `drop: "\".\"`). Removing row indexes that result from the continuation of previous indexes is significantly harder. Hence such rows should be entirely empty (then the row is dropped because it has only `nan` values). It is best to avoid empty index rows and columns altogether.

- When one specifies an open range (north-west corner only) the PandasExcelReader will read from this north-west corner all the way to the end of the sheet (press Ctrl-End in Excel to locate the
cursor into the last cell of a sheet). It does not stop at blank row or column indexes as the tool gdxrw does. This aggravates the situation with empty index cells. Hence the following table will result in the following pandas.DataFrame (use PandasExcelReader attribute trace>2 to print intermediate data frames):

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
<th>j3</th>
<th>Unnamed: 5</th>
<th>Unnamed: 6</th>
<th>Unnamed: 7</th>
<th>Unnamed: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>1.0</td>
<td>2.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i2</td>
<td>NaN</td>
<td>1.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i3</td>
<td>NaN</td>
<td>NaN</td>
<td>1.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>i3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>999.0</td>
</tr>
</tbody>
</table>

which arrives after dropping the nan values and indexes in the Connect database (here a display as a GAMS parameter) as:

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
<th>j3</th>
<th>Unnamed: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i3</td>
<td></td>
<td>1</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

4.46.5.10 PandasExcelWriter

The PandasExcelWriter agent allows to write symbols from the Connect database to an Excel file. Its implementation is based on the pandas.DataFrame class and its I/O API method to_excel. PandasExcelWriter can write sets, parameters, variables, and equations. If the Excel workbook exists, PandasExcelWriter will attempt to append to the existing workbook.

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excelWriterArguments</td>
<td>global</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the pandas.ExcelWriter constructor.</td>
</tr>
<tr>
<td>file</td>
<td>global</td>
<td>None</td>
<td>Specify an Excel file path.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>range</td>
<td>symbol</td>
<td>None</td>
<td>Specify the Excel range of a symbol.</td>
</tr>
<tr>
<td>rowDimension</td>
<td>global/symbol</td>
<td>None</td>
<td>Row dimension of the symbol.</td>
</tr>
<tr>
<td>toExcelArguments</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the pandas.to_excel method.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>valueSubstitutions</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary used for mapping in the value column of the DataFrame.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**excelWriterArguments = dictionary (optional)**

Dictionary containing keyword arguments for the pandas.ExcelWriter constructor.

**file = string (required)**

Specify an Excel file path.

**name = string (required)**
Specify a symbol name for the Connect database.

range = string (required)

Specify the Excel range of a symbol using the format sheet!range. range can be either a single cell (north-west corner like B2) or a full range (north-west and south-east corner like B2:D4). For writing purposes, the south-east corner is ignored. Please note, that PandasExcelWriter always writes an index in the row and column. So even for a scalar or an indexed symbol with rowDimension=0 or dim-rowDimension=0 there will be some index information in the 0 index. For example, the following Connect script

```
parameter x0 / 3.14 /, x1 / i1 1, i2 2, i3 3 /;
$onEmbeddedCode Connect:
- GAMSReader:
  readAll: True
- PandasExcelWriter:
  file: x.xlsx
  symbols:
  - name: x0
    range: Sheet1!a1
  - name: x1
    rowDimension: 1
    range: Sheet1!d1
  - name: x1
    rowDimension: 0
    range: Sheet1!g1
$offEmbeddedCode
```

creates the following Excel output:

rowDimension = integer (optional)

Row dimension: The first rowDimension index positions of the symbol to be written will be written to the rows. dim-rowDimension index positions will be written into the column headers.

toExcelArguments = dictionary (optional)

Dictionary containing keyword arguments for the pandas.DataFrame.to_excel method. Not all arguments of that method are exposed through the YAML interface of the PandasExcelWriter agent. By specifying toExcelArguments, it is possible to pass arguments directly to the pandas.to_excel method that is used by the PandasExcelWriter agent.

trace = integer (optional)

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate data frames will be written abbreviated to the log. For trace > 3 the intermediate data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

valueSubstitutions = dictionary (optional)

Dictionary used for mapping in the value column of the DataFrame. Each key in valueSubstitutions is replaced by its corresponding value. The replacement is only performed on the value column of the DataFrame which is the numerical value in case of a GAMS parameter, variable or equation and the set element text in case of a GAMS set.

4.46.5.11 PandasSQLReader

The PandasSQLReader agent allows to read symbols from a specified database management system into the Connect database. Its implementation is based on the pandas.DataFrame class and its I/O API method read_sql. The method takes in the SQL query or a SQLAlchemy connectable url.
<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>global</td>
<td>None</td>
<td>Connection string to the database.</td>
</tr>
<tr>
<td>dTypeMap</td>
<td>global/symbol</td>
<td>None</td>
<td>Dictionary used to specify the dType of columns.</td>
</tr>
<tr>
<td>indexSubstitutions</td>
<td>symbol</td>
<td>True</td>
<td>Dictionary used for substitutions in the index columns.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>query</td>
<td>symbol</td>
<td>None</td>
<td>Specify the SQL query.</td>
</tr>
<tr>
<td>readSQLArguments</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the pandas.read_sql method.</td>
</tr>
<tr>
<td>sqlEngineArguments</td>
<td>global</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the sqlalchemy.create_engine constructor.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>type</td>
<td>global/symbol</td>
<td>par</td>
<td>Control the symbol type.</td>
</tr>
<tr>
<td>valueColumns</td>
<td>symbol</td>
<td>inferred</td>
<td>Specify columns to get the values from.</td>
</tr>
<tr>
<td>valueSubstitutions</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary used for mapping in the value column of the DataFrame.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**connection = string (required)**

Allows to specify the connection string to the database and user credentials in order to access
the database. Specify the absolute path to the database or specify SQLAlchemy connectable
url. Here are some examples:

Microsoft Access:

connection: "access+pyodbc://?odbc_connect=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=<absolute//path//to//datafile.accdb(.mdb)>;"

SQL Server:

connection: "mssql://?odbc_connect=DRIVER={SQL Server Native Client 11.0};SERVER=<computerName>\SQLEXPRESS;Trusted_Connection=yes;"

SQLite:

connection: "sqlite:///<absolute//path//to//foo.db>"

Postgres:

connection: "postgresql://scott:tiger@localhost/mydatabase"

**dTypeMap = dict (optional)**

Allows to specify the dType of columns in a dictionary as key: value pairs, i.e. <column>:<dtype>.

**indexSubstitutions = dict (optional)**

Dictionary used for substitutions in the index columns. Each key in indexSubstitutions is
replaced by its corresponding value. This option allows arbitrary replacements in the index columns.

**name = string (required)**
Name of the symbol in the Connect database. The name must be unique for each symbol.

**query** = string (required)

Specify the SQL query which will fetch the desired table from the database system.

**readSQLArguments** = dict (optional)

Dictionary containing keyword arguments for the pandas.read_sql method. Not all arguments of that method are exposed through the YAML interface of the PandasSQLReader agent. By specifying readSQLArguments, it is possible to pass arguments directly to the pandas.read_sql method that is used by the PandasSQLReader agent.

**sqlEngineArguments** = dict (optional)

Dictionary containing keyword arguments for the sqlalchemy.create_engine constructor.

**trace** = integer (optional)

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate arrays and data frames will be written abbreviated to the log. For trace > 3 the intermediate arrays and data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

**type** = string (default=par)

Control the symbol type. Supported symbol types are par for GAMS parameters and set for GAMS sets.

**valueColumns** = list or string (optional)

Specify columns to get the values from. The value columns contain numerical values in case of a GAMS parameter and set element text in case of a GAMS set. The columns are given as column names represented as a list of strings. For example: valueColumns: ["i1", "i2"]. If there is more than one value column specified, the column names are stacked to index automatically. As string one can specify the symbolic constant lastCol (i.e. valueColumns: "lastCol") or an empty string (i.e. valueColumns: "). When lastCol is passed the last column will be treated as a value column and all the other columns will be treated as index columns. When an empty list is passed all columns will be treated as index columns. Specifying an empty list is only valid for symbol type set since symbol type par requires at least one value column. The default for symbol type par is lastCol and the default for symbol type set is an empty string.

**valueSubstitutions** = dict (optional)

Dictionary used for mapping in the value column of the DataFrame. Each key in valueSubstitutions is replaced by its corresponding value. The replacement is only performed on the value column of the DataFrame which is the numerical value in case of a GAMS parameter and the set element text in case of a GAMS set. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading.

### 4.46.5.12 PandasSQLWriter

The PandasSQLWriter agent allows to write symbols from the Connect database to a specified database management system. Its implementation is based on the pandas.DataFrame class and its I/O API method to_sql.
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>global</td>
<td>None</td>
<td>Connection string to the database.</td>
</tr>
<tr>
<td>ifExists</td>
<td>symbol</td>
<td>fail</td>
<td>Specify the behavior when a table with the same name exists in the database/schema.</td>
</tr>
<tr>
<td>name</td>
<td>symbol</td>
<td>None</td>
<td>Specify the name of the symbol in the Connect database.</td>
</tr>
<tr>
<td>schemaName</td>
<td>global/symbol</td>
<td>None</td>
<td>Specify the schema name.</td>
</tr>
<tr>
<td>sqlEngineArguments</td>
<td>global</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the <code>sqlalchemy.create_engine</code> constructor.</td>
</tr>
<tr>
<td>tableName</td>
<td>symbol</td>
<td>None</td>
<td>Specify the SQL table/relation in the provided database/schema.</td>
</tr>
<tr>
<td>toSQLArguments</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary containing keyword arguments for the <code>pandas.to_sql</code> method.</td>
</tr>
<tr>
<td>trace</td>
<td>global</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>unstack</td>
<td>symbol</td>
<td>False</td>
<td>Indicate if the last index column will be used as a header row.</td>
</tr>
<tr>
<td>valueSubstitutions</td>
<td>symbol</td>
<td>None</td>
<td>Dictionary used for mapping in the value column of the <code>DataFrame</code>.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**connection = string (required)**

Allows to specify the connection string to the database and user credentials in order to access the database. Specify the absolute path to the database or specify SQLAlchemy connectable url.

**ifExists = string (default=fail)**

Specify the behavior when a table with the same name exists in the database/schema. Valid values are `fail`, `replace` and `append`.

**name = string (required)**

Specify the name of the symbol in the Connect database.

**schemaName = string (optional)**

Specify the schema name for writing the table to the correct location. In Postgres, by default, it writes to a public schema already present in every database.

**sqlEngineArguments = dict (optional)**

Dictionary containing keyword arguments for the `sqlalchemy.create_engine` constructor.

**tableName = string (required)**

Name of the SQL table/relation in the provided database/schema.

**toSQLArguments = dict (optional)**

Dictionary containing keyword arguments for the `pandas.to_sql` method. Not all arguments of that method are exposed through the YAML interface of the PandasSQLWriter agent. By specifying `toSQLArguments`, it is possible to pass arguments directly to the `pandas.to_sql` method that is used by the PandasSQLWriter agent.
**trace** = integer (optional)

Specify the trace level for debugging output. For trace > 1 some scalar debugging output will be written to the log. For trace > 2 the intermediate arrays and data frames will be written abbreviated to the log. For trace > 3 the intermediate arrays and data frames will be written entirely to the log (potentially large output). If trace has not been set, the trace value, set by the Options agent, will be used.

**unstack** = boolean (default=False)

Indicate if the last index column will be used as a header row.

**valueSubstitutions** = dict (optional)

Dictionary used for mapping in the value column of the DataFrame. Each key in valueSubstitutions is replaced by its corresponding value. The replacement is only performed on the value column of the DataFrame which is the numerical value in case of a GAMS parameter, variable or equation and the set element text in case of a GAMS set.

### 4.46.5.13 Projection

The Projection agent allows index reordering and projection onto a reduced index space of a GAMS symbol. For variables and equations a suffix (\(^\cdot\.l\), \(^\cdot\.m\), \(^\cdot\.lo\), \(^\cdot\.up\), or \(^\cdot\.scale\)) can be extracted and is written to a parameter. Otherwise, the type of the source symbol determines the type of the new symbol, unless asSet is set to True. Variables and equations can be turned into parameters with an extra index containing the labels level, marginal, lower, upper, and scale if the attribute asParameter is set to True. Moreover, if name is a list of scalar symbols of the same type (parameters, variables, or equations), they can be stored in a one-dimensional symbols of the same type with the index label being the name of the scalar symbol.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregationMethod</td>
<td>first</td>
<td>Specify the aggregation method for the projection.</td>
</tr>
<tr>
<td>asParameter</td>
<td>False</td>
<td>Indicate that variable or equation symbols will be turned into a parameter with an extra index that contains the suffix.</td>
</tr>
<tr>
<td>asSet</td>
<td>False</td>
<td>Indicate that the new symbol is a set independent of the type of the source symbol.</td>
</tr>
<tr>
<td>name</td>
<td>None</td>
<td>Specify a symbol name with index space and potentially suffix for the Connect database.</td>
</tr>
<tr>
<td>newName</td>
<td>None</td>
<td>Specify a new name with index space for the symbol in the Connect database.</td>
</tr>
<tr>
<td>text</td>
<td>None</td>
<td>Element text for resulting sets.</td>
</tr>
<tr>
<td>trace</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**aggregationMethod** = string (default=first)

Specify the method to aggregate when at least one index position is projected out. The default is first, meaning that the first record will be stored in the new symbol. For sets, variables, and equations (without a suffix specified) only first and last are meaningful. For parameters, variables, and equations with suffix many other aggregation methods are
available and meaningful: \texttt{mad} (mean absolute deviation), \texttt{max}, \texttt{mean}, \texttt{median}, \texttt{min}, \texttt{prod}, \texttt{sem} (unbiased standard error of the mean), \texttt{sum}, \texttt{std} (standard deviation), \texttt{nunique} (number of distinct elements), \texttt{first}, \texttt{last}. The projection agent is based on pandas DataFrames and more detailed explanations of the aggregation method can be found at the pandas website.

\texttt{asParameter = boolean} (default=\texttt{False})

If the symbol specified by \texttt{name} is a variable or equation and \texttt{asParameter} is set to \texttt{True}, the new symbol (after potential \texttt{aggregationMethod} is applied) will be a parameter with an additional index (at the end of the index list) that contains the suffix labels \texttt{level}, \texttt{marginal}, \texttt{lower}, \texttt{upper}, and \texttt{scale} and the corresponding suffix value.

\texttt{asSet = boolean} (default=\texttt{False})

Usually the type of the source symbol and the use of a suffix with variables and equations determine the type of the target symbol. With \texttt{asSet} set to \texttt{True} the target symbol will be a set.

\texttt{name = string or list of strings} (required)

One either specifies a single symbol name with index space and potentially suffix for the Connect database or a list of symbol names of scalar symbols. In the prior case, \texttt{name} requires the format \texttt{symname[.suffix](i1,i2,...,iN)}. The suffix is only allowed on variable and equation symbols and need to be either \texttt{l}, \texttt{m}, \texttt{lo}, \texttt{up}, or \texttt{scale}. The list of indices does not need to coincide with the names of the actual GAMS domain sets. This index list together with the index list specified for \texttt{newName} is solely intended to establish the index order in the symbol specified by \texttt{newName}. In the latter case (a list of symbol names) the symbols need to be scalar symbols of the same type (parameter, variable, or equation) and a new one-dimensional symbol (of the same type) is created (using \texttt{newName}) that holds the symbol names as labels.

\texttt{newName = string} (required)

Specify a new name with index space for the projected or reordered symbol in the Connect database. Note that each symbol in the Connect database must have a unique name. \texttt{newName} is given as \texttt{symname(i1,i2,...,iN)}. The list of indices does not need to coincide with the names of the actual GAMS domain sets. This index list together with the index list specified for \texttt{name} is solely intended to establish the index order. Hence, the names in the index list need to be unique and only names that are part of the index list specified for \texttt{name} can be used. For example: \texttt{name: p(I,j,k)} and \texttt{newName: q(k,i)}.

\texttt{text = string} (default=\texttt{None})

Control the handling of element text if the resulting symbol is a set. If set to "", the text will be dropped. When left at default (\texttt{None}) and the projected symbol is a set, the element text of the original set will be used. For other symbols types, the text will be dropped. If \texttt{text} is a string, this string will be assigned to all elements. The string can contain \texttt{place holders} \{i1\} that will be replaced with the content of the matching index position. For example, \texttt{text: \"{i2} - {i1}: \{element_text\}\", where \{i1\} and \{i2\} should be index space names in the symbol name with index space (attribute \texttt{name}). \{element_text\} refers to original element text (set) or string representation of a numerical value (parameter or variable/equation with a given suffix) of the source symbol.

\texttt{trace = integer} (optional)

Specify the trace level for debugging output. For \texttt{trace} \textgreater 1 some scalar debugging output will be written to the log. For \texttt{trace} \textgreater 2 the intermediate arrays and data frames will be written abbreviated to the log. For \texttt{trace} \textgreater 3 the intermediate arrays and data frames will be written entirely to the log (potentially large output). If \texttt{trace} has not been set, the \texttt{trace} value, set by the \texttt{Options} agent, will be used.
4.46.14 PythonCode

The PythonCode agent allows to execute arbitrary Python code. From within the code, it is possible to access the GAMS database via `gams.db` (if the PythonCode agent is running in a GAMS context) and the Connect database via `connect.container`. The GAMS database is an instance of `GamsDatabase`, whereas the Connect database is a `GAMS Transfer Container` object. Furthermore there is a predefined `instructions` list that can be filled with tasks that are automatically executed.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>None</td>
<td>Python code to be executed.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

code = string (required)

Python code to be executed. The YAML syntax offers the pipe character (|) for specifying multi-line strings:

- PythonCode:
  
  code: |
  print("Print from Python")
  # insert more Python code here

It is possible to generate instructions by appending tasks to the Python `instructions` list. A task is specified by using Python data structures that match the schema of a specific Connect agent. At the end of the Python code, all tasks in the `instructions` list are automatically generated and executed. The following example shows how to fill the `instructions` list with three PandasExcelWriter tasks that write different parameters (p0, p1, p2) into separate Excel workbooks (`data_p0.xlsx`, `data_p1.xlsx`, `data_p2.xlsx`).

- GAMSReader:
  
  readAll: True

- PythonCode:
  
  code: |
  symbols = ['p0', 'p1', 'p2']
  for s in symbols:
    instructions.append(
      {'PandasExcelWriter':
       {'file': 'data_{}.xlsx'.format(s),
        'symbols': [{'name': s, 'rowDimension': connect.container.data[s].dimension, 'range': s+'!A1'}]
      })

Using `gams.db` and `connect.container` allows to access symbols directly. The following example shows how to manually fill a symbol in the GAMS database (`gams.db`) with a symbol from the Connect database (`connect.container`) which would normally be done using the `GAMSWriter` agent.

- GAMSReader:
  
  readAll: True

- PythonCode:
  
  code: |
  connect.container.data['p1'].copy_symbol(gams.db['p2'])
4.46.5.15 RawExcelReader

The RawExcelReader allows reading of unstructured data from a specified Excel file into the Connect database. This reader works similarly compared to the xlsdump tool. It reads the entire spreadsheet and represents its content in a couple of GAMS sets:

- s /s1, s2,.../ (workbook sheets)
- w / Sheet1, Sheet2, ... / (workbook sheets by name)
- ws(s,w) / s1.Sheet1, s2.Sheet2, ... / (workbook map)
- r / r1, r2, ... / (rows)
- c / c1, c2, ... / (columns)
- vs(s,r,c) / s1.r1.c2 "cell text", ... / (cells with explanatory text)
- vu(s,r,c,* / s1.r1.c1."cell text" "cell text", ... (cells with potential GAMS label)

and a parameter vf(s,r,c) / s1.r2.c2 3.14, ... / (cells with numerical values). Cells with a date will be stored in it's string representation in vu and as a Julian date in vf. Cells with a string value will be stored in vs. If the string length exceeds the maximum length allowed for elements text, it will be truncated. Excel offers many other cell value types. RawExcelReader will try to represent the cell value as a number and if this succeeds stores the number in vf. Strings of GAMS special values names INF, EPS, NA, and UNDEF will be also converted to its numerical counterpart. It will also try to represent the cell value as a string and stores this as a label in the fourth position in vu. GAMS labels have a length limitation and hence RawExcelReader automatically shortens the label to fit this limit. RawExcelReader will provide a unique label (ending in ∼n where n is an integer for strings exceeding the label length limit) for each string in the workbook. The full string (if it fits) will be available as the element text of the vu record.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>c</td>
<td>Symbol name for columns.</td>
</tr>
<tr>
<td>columnLabel</td>
<td>C</td>
<td>Label for columns.</td>
</tr>
<tr>
<td>file</td>
<td>None</td>
<td>Specify an Excel file path.</td>
</tr>
<tr>
<td>mergedCells</td>
<td>False</td>
<td>Control the handling of empty cells that are part of a merged Excel range.</td>
</tr>
<tr>
<td>rName</td>
<td>r</td>
<td>Symbol name for rows.</td>
</tr>
<tr>
<td>rowLabel</td>
<td>R</td>
<td>Label for rows.</td>
</tr>
<tr>
<td>sheetLabel</td>
<td>S</td>
<td>Label for workbook sheets.</td>
</tr>
<tr>
<td>sName</td>
<td>s</td>
<td>Symbol name for workbook sheets.</td>
</tr>
<tr>
<td>trace</td>
<td>inherited</td>
<td>Specify the trace level for debugging output.</td>
</tr>
<tr>
<td>vfName</td>
<td>vf</td>
<td>Symbol name for cells with a numerical value.</td>
</tr>
<tr>
<td>vsName</td>
<td>vs</td>
<td>Symbol name for cells with an explanatory text.</td>
</tr>
<tr>
<td>vuName</td>
<td>vu</td>
<td>Symbol name for cells with a potential GAMS label.</td>
</tr>
<tr>
<td>wName</td>
<td>w</td>
<td>Symbol name for workbook sheets by name.</td>
</tr>
<tr>
<td>wsName</td>
<td>w</td>
<td>Symbol name for workbook map.</td>
</tr>
</tbody>
</table>

Detailed description of the options:

**cName = string (default=c)**

Control the name of the set of columns.

**columnLabel = string (default=C)**
Control the labels for the set of columns (c).

**file = string** (required)

Specify an Excel file path.

**mergedCells = boolean** (default=\texttt{False})

Control the handling of empty cells that are part of a merged Excel range. If \texttt{False}, the cells are left empty. If \texttt{True}, the merged value is used in all cells. Note that setting this option to \texttt{True} has an impact on performance since the Excel workbook has to be opened in a non-read-only mode that results in non-constant memory consumption (no lazy loading).

**rName = string** (default=\texttt{r})

Control the name of the set of rows.

**rowLabel = string** (default=\texttt{R})

Control the labels for the set of rows (r).

**sheetLabel = string** (default=\texttt{S})

Control the labels for the set of workbook sheet (s).

**sName = string** (default=\texttt{s})

Control the name of the set of workbook sheets.

**trace = integer** (optional)

Specify the trace level for debugging output. For \texttt{trace} > 1 some scalar debugging output will be written to the log. For \texttt{trace} > 2 the cell values and its processing will be written entirely to the log (potentially large output). If \texttt{trace} has not been set, the \texttt{trace} value, set by the \texttt{Options} agent, will be used.

**vfName = string** (default=\texttt{vf})

Control the name of the parameter for cells with a numerical value.

**vsName = string** (default=\texttt{vs})

Control the name of the set for cells with an explanatory text.

**vuName = string** (default=\texttt{vu})

Control the name of the set for cells with a potential GAMS label.

**wName = string** (default=\texttt{w})

Control the name of the set of workbook sheets by name.

**wsName = string** (default=\texttt{ws})

Control the name of the set of the workbook map.
4.46.6 Text Substitutions in YAML File

In many cases one would like to parameterize the text in the Connect instruction file. For example, some of the Connect agents require a file name. Instead of hard coding the file name into the YAML instructions, text substitutions allow to have a place holder for the attribute that is substituted out before giving the instructions to Connect. The place holder in the YAML file uses the syntax `%SOMETEXT%`, similar to the GAMS compile-time variables. For example:

- CSVReader:
  
  file: %MYFILENAME%
  name: distance
  indexColumns: [1, 2]
  valueColumns: [3]

Depending on how Connect runs, the substitution is done in various ways. The section Substitutions in Embedded Connect Code described the substitution mechanisms for embedded Connect code. When Connect is initiated via the command line parameters connectIn or connectOut, the user defined parameter specified by double-dash command line parameters and the given GAMS command line parameters, e.g. `%gams.input%` will be substituted in the YAML file. The list of parameters available for substitution is printed to the GAMS log at the beginning of the job in the section GAMS Parameters defined.

When Connect is initiated via the shell command `gamsconnect` all substitutions need to be specified on the command line:

```
gamsconnect myci.yaml key1=val1 key2=val2 ...
```

`key` can be just `MYFILENAME` or be composed like `gams.Input` or `system.dirSep`.

4.46.7 Use Connect Agents in Custom Python Code

Instead of passing instructions via one of the Connect interfaces, users can execute tasks directly in their Python code by creating an instance of `ConnectDatabase` and calling method `.exec_task(task)`. The `task` argument is expected to be a Python dictionary of form:

```python
{
    '<agent name>':,
    {
        '<global option1>': <value>,
        '<global option2>': <value>,
        ...
        'symbols':
        [
            {<symbol1 option1>':<value>,
             '<symbol1 option2>': <value>,
             '
             ...
            },
            {<symbol2 option1>':<value>,
             '<symbol2 option2>': <value>,
             ...
            },
            ...
        ]
    }
}
```
Users can either construct the Python dictionary themselves or let YAML create the dictionary from a YAML script. The following example creates an instance of ConnectDatabase and executes two tasks: First, the CSV file `stockprice.csv` is read into the Connect database and second, the symbol `stockprice` is written to the GAMS database. In this example, the tasks are directly specified as Python dictionaries.

```
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onEcho > stockprice.csv
date;symbol;price
2016/01/04;AAPL;105,35
2016/01/04;AXP;67,59
2016/01/04;BA;140,50
$offEcho

$onEmbeddedCode Python:
from gams_connect.connectdatabase import ConnectDatabase
cdb = ConnectDatabase(gams._system_directory, gams)
cdb.exec_task({'CSVReader': {'file': 'stockprice.csv', 'name': 'stockprice', 'indexColumns': [1, 2], 'valueColumns': [3], 'fieldSeparator': ';', 'decimalSeparator': ','}})
cdb.exec_task({'GAMSWriter': {'symbols': [{'name': 'stockprice'}]}})
$offEmbeddedCode

display stockprice;
```

We can also construct the Python dictionaries by using YAML:

```
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onEcho > stockprice.csv
date;symbol;price
2016/01/04;AAPL;105,35
2016/01/04;AXP;67,59
2016/01/04;BA;140,50
$offEcho

$onEmbeddedCode Python:
import yaml
from gams_connect.connectdatabase import ConnectDatabase
cdb = ConnectDatabase(gams._system_directory, gams)
inst = yaml.safe_load(''
- CSVReader:
  - file: stockprice.csv
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
    fieldSeparator: ';'
    decimalSeparator: ','
- GAMSWriter:
  - name: stockprice
''
)
for task in inst:
    cdb.exec_Task(task)
```

Here YAML creates a list of dictionaries (i.e. a list of tasks) from the given YAML script.

### 4.46.8 Command Line Utility gamsconnect

The GAMS system directory contains the utility `gamsconnect` to run Connect instructions directly from the command line. On Windows the utility has the callable extension `.cmd`. This script wraps the Python script `connectdriver.py` by calling the Python interpreter that ships with GAMS. `gamsconnect` operates as the other Connect drivers on a YAML instruction file. The agents `GAMSReader` and `GAMSWriter` are not available from `gamsconnect` and will trigger an exception. Substitutions can be passed to `gamsconnect` via command line arguments as `key=value`, e.g. `filename=myfile.csv` and even `gams.scrdir=/tmp/`. `gamsconnect` is called like this:

```
gamsconnect <YAMLFile> [key1=value1 [key2=value2 [key3=value3 [...]]]]
```

### 4.47 Extrinsic Functions

#### 4.47.1 Introduction

Mathematical functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or intrinsic functions. GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, since it allows users to import functions from an external library into a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first and second derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with extrinsic functions. This includes the class of deterministic global solvers, see column "Global" in this table, while, for example, stochastic global solvers can work with extrinsic functions.

In this chapter we will demonstrate how to access functions from an extrinsic function library in a GAMS model and we will describe the extrinsic function libraries that are included in the GAMS distribution. In addition, we will provide some pointers for users who wish to build their own extrinsic function library.

#### 4.47.2 Using Function Libraries

Function libraries are made available to a model with the following compiler directive:

```
$funcLibIn <InternalLibName> <ExternalLibName>
```

Here `InternalLibName` is a handle that will be used to refer to the library inside the model source code. `ExternalLibName` is the file name of the shared library that implements the extrinsic functions. If no path is given, GAMS will look for the library in the directory `extrinsic_functions` in the GAMS standard locations and in the GAMS system directory. To access a library that does not reside in these standard places, the external name should include a relative or absolute path to the location of the library. GAMS will then search for the specified library using the mechanisms specific to the host operating system. When processing the directive `$funcLibIn`, GAMS will validate the library, make the included functions available for use and add a table of the included functions to the listing file.
Note

The function library facility gives users complete control over naming so that potential name conflicts between libraries can be avoided.

Before the individual functions may be used, they have to be declared in the following way:

Function \(<\text{InternalFuncName}> /\langle\text{InternalLibName}\rangle.<\text{FuncName}>;/

Here \text{InternalFuncName} is the name of the individual function that will be used in the GAMS code. The user may choose this internal name freely and thus avoid potential naming conflicts. \text{InternalLibName} is the name of the function library as defined by the \$\text{funcLibIn} directive and \text{FuncName} is the name of the individual function in the external function library. Once functions have been declared in this way they may be used like intrinsic functions.

Consider the following simple example:

\$\text{funcLibIn myLib tricclib}

Function myCos /myLib.Cosine/
    mySin /myLib.Sine/
    myPi /myLib.Pi/;

Scalar d;
  d = myCos(myPi/3);
  display d;

Note that in the first line the external trigonometric library \text{tricclib} is activated and the internal name \text{myLib} is specified for it. Then the functions are declared. Observe that \text{Cosine}, \text{Sine} and \text{Pi} are functions in the trigonometric library. After the library has been loaded and the functions have been declared, the functions may be used as usual. The trigonometric library is discussed in section \text{Example: Trigonometric Library} below.

4.47.3 Libraries that are included in the GAMS Distribution

In this section we will present the libraries that are included in the GAMS distribution: the \text{Fitpack Library}, the \text{Piecewise Polynomial Library}, the \text{Stochastic Library} and the \text{LINDO Sampling Library}. Note that the LINDO Sampling Library is only available for LINDO license holders. In addition, we will provide details on the \text{Mutex Library}.

In the tables that follow, the "Endogenous Classification" (last column) specifies in which models the function may legally appear. In order of least to most restrictive, the choices are \text{any}, \text{DNLP}, \text{NLP}, \text{none}. See section \text{Classification of Models} for details on model types in GAMS. Note well that functions classified as \text{any} are only permitted with exogenous (constant) arguments.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogenous variables, see section \text{Functions in Equation Definitions}. Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.
4.47.3.1 The Fitpack Library

FITPACK by Paul Dierckx [43] is a Fortran-based library for one-dimensional and two-dimensional spline interpolations. This library has been repackaged to work with the GAMS Function Library Facility. The model [FITLIB01] from the GAMS Test Library is an example of the use of the library FITPACK inside GAMS.

Note that the supporting points to which the function will be fit need to be stored in a three-dimensional parameter, fitdata, in a GDX file fit.gdx. The first dimension is a function index, the second dimension is the index of the supporting point and the third dimension takes one of the following four values: "w" (weight), "x" (x-value), "y" (y-value) or "z" (z-value).

The FITPACK library is made available with the following directive:

$funcLibIn <InternalLibName> fitfclib

It provides the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitFunc(FUNIND,x[,y])</td>
<td>Evaluate spline</td>
<td>DNLP</td>
</tr>
<tr>
<td>fitParam(FUNIND,PARAM[,VALUE])</td>
<td>Read or set parameters</td>
<td>none</td>
</tr>
</tbody>
</table>

The function FitParam may be used to change certain parameters that are used for the evaluation. The following values are defined:

- 1: Smoothing factor (S)
- 2: Degree of spline in direction x (Kx)
- 3: Degree of spline in direction y (Ky)
- 4: Lower bound of function in direction x (LOx)
- 5: Lower bound of function in direction y (LOY)
- 6: Upper bound of function in direction x (UPx)
- 7: Upper bound of function in direction y (UPy)

4.47.3.2 The Piecewise Polynomial Library

The Piecewise Polynomial Library may be used to evaluate piecewise polynomial functions. An example is given in the model [PWPLIB01] in the GAMS Test Library. Note that the functions that are to be evaluated need to be defined and stored in a GDX file. The following code snippet serves as illustration:

* Define two piecewise polynomial functions

Table pepdata(*,* ,*) '1st index: function number, 2nd index: segment number, 3rd index: degree'
<table>
<thead>
<tr>
<th>function</th>
<th>segment</th>
<th>degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>leftBound</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>2.4</td>
<td>-2.7</td>
</tr>
<tr>
<td>1.2</td>
<td>5.6</td>
<td>-4.3</td>
</tr>
<tr>
<td>2.1</td>
<td>0</td>
<td>-6.3333</td>
</tr>
<tr>
<td>2.2</td>
<td>0.3333</td>
<td>1.0370</td>
</tr>
</tbody>
</table>
Write pwp data to gdx file, which will be read by external library (pwpcclib)

```
.gdxout pwp.gdx
$unload pwpdata
.gdxout
```

Observe that on each row of the table `pwpdata` we have the following entries:

- `FuncInd` sets a function index
- `SegInd` defines the index of the segment (or interval) which is described
- `leftBound` gives the lower bound of the segment
- `CoefX` defines the X-th degree coefficient of the polynomial corresponding to this segment

The Piecewise Polynomial Library is made available with the following directive:

```
$funcLibIn <InternalLibName> pwpcclib
```

It provides the following function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwpFunc(FUNIND,x)</td>
<td>Piecewise polynomials</td>
<td>DNLP</td>
</tr>
</tbody>
</table>

### 4.47.3.3 The Stochastic Library

The Stochastic Library provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain continuous and discrete distributions. This library is made available with the following directive:

```
$funcLibIn <InternalLibName> stodclib
```

The continuous distributions that are available with this library are the following:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta(SHAPE_1,SHAPE_2)</td>
<td>Beta distribution with shape parameters SHAPE_1 and SHAPE_2, see MathWorld</td>
</tr>
<tr>
<td>cauchy(LOCATION,SCALE)</td>
<td>Cauchy distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>ChiSquare(DF)</td>
<td>Chi-squared distribution with degrees of freedom DF, see MathWorld</td>
</tr>
<tr>
<td>exponential(LAMBDA)</td>
<td>Exponential distribution with rate of change LAMBDA, see MathWorld</td>
</tr>
<tr>
<td>f(DF_1,DF_2)</td>
<td>F-distribution with degrees of freedom DF_1 and DF_2, see MathWorld</td>
</tr>
</tbody>
</table>
### Distribution Description

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma(SHAPE, SCALE)</td>
<td>Gamma distribution with shape parameter SHAPE and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>gumbel(LOCATION, SCALE)</td>
<td>Gumbel distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>invGaussian(MEAN, SHAPE)</td>
<td>Inverse Gaussian distribution with mean MEAN and scaling parameter SHAPE, see MathWorld</td>
</tr>
<tr>
<td>laplace(MEAN, SCALE)</td>
<td>Laplace distribution with mean MEAN and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>logistic(LOCATION, SCALE)</td>
<td>Logistic distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>logNormal(LOCATION, SCALE)</td>
<td>Lognormal distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld</td>
</tr>
<tr>
<td>normal(MEAN, STD_DEV)</td>
<td>Normal distribution with mean MEAN and standard deviation STD_DEV, see MathWorld</td>
</tr>
<tr>
<td>pareto(SCALE, SHAPE)</td>
<td>Pareto distribution with scaling parameter SCALE and shape parameter SHAPE, see MathWorld</td>
</tr>
<tr>
<td>rayleigh(SIGMA)</td>
<td>Rayleigh distribution with parameter SIGMA, see MathWorld</td>
</tr>
<tr>
<td>studentT(DF)</td>
<td>Student’s t-distribution with degrees of freedom DF, see MathWorld</td>
</tr>
<tr>
<td>triangular(LOW, MID, HIGH)</td>
<td>Triangular distribution between LOW and HIGH, where MID is the most probable number, see MathWorld</td>
</tr>
<tr>
<td>uniform(LOW, HIGH)</td>
<td>Uniform distribution between LOW and HIGH, see MathWorld</td>
</tr>
<tr>
<td>weibull(SHAPE, SCALE)</td>
<td>Weibull distribution with shape parameter SHAPE and scaling parameter SCALE, see MathWorld</td>
</tr>
</tbody>
</table>

Further, the following discrete distributions are available:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binomial(N,P)</td>
<td>Binomial distribution with number of trials N and success probability P in each trial, see MathWorld</td>
</tr>
<tr>
<td>geometric(P)</td>
<td>Geometric distribution with success probability P in each trial, see MathWorld</td>
</tr>
<tr>
<td>hyperGeo(TOTAL,GOOD,TRIALS)</td>
<td>Hypergeometric distribution with total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS, see MathWorld</td>
</tr>
<tr>
<td>logarithmic(P-FACTOR)</td>
<td>Logarithmic distribution with parameter P-FACTOR, also called log-series distribution, see MathWorld</td>
</tr>
<tr>
<td>negBinomial(FAILURES,P)</td>
<td>Negative Binomial distribution with FAILURES being the number of failures until the experiment is stopped and success probability P in each trial. The generated random number describes the number of successes until the defined number of failures is reached, see MathWorld</td>
</tr>
<tr>
<td>poisson(LAMBDA)</td>
<td>Poisson distribution with mean LAMBDA, see MathWorld</td>
</tr>
<tr>
<td>uniformInt(LOW,HIGH)</td>
<td>Integer Uniform distribution between LOW and HIGH, see MathWorld</td>
</tr>
</tbody>
</table>

Note that for each distribution the library offers the following four functions, where DistributionName is the name of the distribution as listed in the tables above, parameters are the parameters associated with each distribution, and x is the point at which the function is to be evaluated. Note that x may be an endogenous variable.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif. for Continuous Distributions</th>
<th>End. Classif. for Discrete Distributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>d&lt;DistributionName&gt; &lt;parameters&gt;</td>
<td>Generate a random deviate (sample from the distribution)</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>pdf&lt;DistributionName&gt; &lt;parameters&gt;</td>
<td>Probability density function</td>
<td>DNLP</td>
<td>none</td>
</tr>
<tr>
<td>cdf&lt;DistributionName&gt; &lt;parameters&gt;</td>
<td>Cumulative distribution function</td>
<td>DNLP</td>
<td>none</td>
</tr>
<tr>
<td>icdf&lt;DistributionName&gt; &lt;parameters&gt;</td>
<td>Inverse cumulative distribution function</td>
<td>DNLP</td>
<td>none</td>
</tr>
</tbody>
</table>

For example, the functions for the Normal distribution are

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dNormal(MEAN, STD_DEV)</td>
<td>Samples a random number from the Normal distribution</td>
<td>none</td>
</tr>
<tr>
<td>pdfNormal(x, MEAN, STD_DEV)</td>
<td>Probability density function for Normal distribution</td>
<td>DNLP</td>
</tr>
<tr>
<td>cdfNormal(x, MEAN, STD_DEV)</td>
<td>Cumulative distribution function for Normal distribution</td>
<td>DNLP</td>
</tr>
<tr>
<td>icdfNormal(x, MEAN, STD_DEV)</td>
<td>Inverse cumulative distribution function for Normal distribution</td>
<td>DNLP</td>
</tr>
</tbody>
</table>

Finally, the seed for the various random number generators can be set by using the following function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetSeed(SEED)</td>
<td>Defines the seed for random number generator</td>
<td>none</td>
</tr>
</tbody>
</table>

In the following example, a sample of size 20 is generated from the Normal, Binomial, Cauchy, and Lognormal distributions each:

```gams
$funcLibIn stolib stodclib
Functions randnorm /stolib.dnormal /
    randbin /stolib.dbinomial /
    randcauchy /stolib.dcauchy /
    randlognorm /stolib.dlognormal /

Set i /i1*i20/;
Set j /norm, binomial, cauchy, lognorm/;
Parameter randx(i,j) "distribution sample";

randx(i,"norm") = randnorm(5,2);
randx(i,"binomial") = randbin(10,0.5);
randx(i,"cauchy") = randcauchy(5,1);
randx(i,"lognorm") = randlognorm(1.2,0.3);
display randx;
```

In the example, first the stochastic library is made available in GAMS, then the functions that will be used from the library are declared, giving them names under which to refer to them in the GAMS model. The output generated by the display statement is the following:
4.47 Extrinsic Functions

---

**PARAMETER randx distribution sample**

<table>
<thead>
<tr>
<th></th>
<th>norm</th>
<th>binomial</th>
<th>cauchy</th>
<th>lognorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>4.373</td>
<td>4.000</td>
<td>5.520</td>
<td>3.132</td>
</tr>
<tr>
<td>i2</td>
<td>5.655</td>
<td>6.000</td>
<td>6.813</td>
<td>4.192</td>
</tr>
<tr>
<td>i3</td>
<td>5.927</td>
<td>6.000</td>
<td>5.426</td>
<td>2.801</td>
</tr>
<tr>
<td>i4</td>
<td>3.340</td>
<td>4.000</td>
<td>5.898</td>
<td>3.689</td>
</tr>
<tr>
<td>i5</td>
<td>3.537</td>
<td>3.000</td>
<td>-3.069</td>
<td>2.746</td>
</tr>
<tr>
<td>i6</td>
<td>3.057</td>
<td>5.000</td>
<td>5.518</td>
<td>3.430</td>
</tr>
<tr>
<td>i7</td>
<td>4.212</td>
<td>3.000</td>
<td>0.136</td>
<td>1.577</td>
</tr>
<tr>
<td>i8</td>
<td>6.869</td>
<td>7.000</td>
<td>5.068</td>
<td>3.857</td>
</tr>
<tr>
<td>i9</td>
<td>3.481</td>
<td>4.000</td>
<td>0.136</td>
<td>1.577</td>
</tr>
<tr>
<td>i10</td>
<td>5.001</td>
<td>4.000</td>
<td>5.274</td>
<td>2.261</td>
</tr>
<tr>
<td>i11</td>
<td>3.182</td>
<td>5.000</td>
<td>4.383</td>
<td>2.686</td>
</tr>
<tr>
<td>i12</td>
<td>5.688</td>
<td>6.000</td>
<td>2.914</td>
<td>2.102</td>
</tr>
<tr>
<td>i13</td>
<td>3.675</td>
<td>6.000</td>
<td>4.693</td>
<td>3.105</td>
</tr>
<tr>
<td>i14</td>
<td>4.028</td>
<td>5.000</td>
<td>1.813</td>
<td>2.418</td>
</tr>
<tr>
<td>i15</td>
<td>8.767</td>
<td>5.000</td>
<td>12.190</td>
<td>2.182</td>
</tr>
<tr>
<td>i16</td>
<td>3.558</td>
<td>3.000</td>
<td>-112.644</td>
<td>2.092</td>
</tr>
<tr>
<td>i17</td>
<td>2.402</td>
<td>4.000</td>
<td>4.078</td>
<td>2.523</td>
</tr>
<tr>
<td>i18</td>
<td>2.249</td>
<td>2.000</td>
<td>0.996</td>
<td>2.777</td>
</tr>
<tr>
<td>i19</td>
<td>5.639</td>
<td>4.000</td>
<td>3.931</td>
<td>2.678</td>
</tr>
<tr>
<td>i20</td>
<td>7.374</td>
<td>4.000</td>
<td>4.671</td>
<td>3.159</td>
</tr>
</tbody>
</table>

4.47.3.4 The LINDO Sampling Library

The LINDO Sampling Library provides samples of random numbers for certain distributions.

It is made available by the following directive:

$\text{funcLibIn InternalLibName lsadclib}$

Observe that a LINDO license is required to use this library.

The following table list the LINDO sampling functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End.</th>
<th>Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sampleLS &lt;DistributionName&gt;</td>
<td>Creates a sample using the distribution &lt;DistributionName&gt;, according to the distribution parameters, and returns a HANDLE that references the sample, as illustrated in the example below.</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>getSampleValues(HANDLE)</td>
<td>Retrieves sampling created by the function sampleLS. See example below.</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>induceCorrelation(CORTYPE)</td>
<td>Induces the correlation that was set with the function setCorrelation before. CORTYPE describes the correlation type: 0 (Pearson), 1 (Kendall) or 2 (Spearman). See example below.</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>setCorrelation(SAMPLE1,SAMPLE2,COR)</td>
<td>Defines correlation between two samplings. See example below.</td>
<td></td>
<td>none</td>
</tr>
</tbody>
</table>
setSeed(SEED)

Specifies the seed for the random number generator.

setRNG(RNG)

Specifies the random number generator that will be used. Possible values are
-1 (FREE), 0 (SYSTEM), 1 (LINDO1), 2 (LINDO2), 3 (LIN1), 4 (MULT1), 5 (MULT2), and 6 (MERSENNE).

The following tables list the available continuous and discrete distributions, respectively. Note that the parameter SAMSIZE must be specified and describes the size of the sample. However, the parameter VARRED is optional and facilitates choosing a variance reduction method. The values are 0 (meaning "none"), 1 (meaning "Latin Hyper Square") and 2 (meaning "Antithetic"). The default is Latin Hyper Square sampling, it will be used if no variance reduction method is specified.

<table>
<thead>
<tr>
<th>Continuous Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta(SHAPE_1,SHAPE_2,SAMSIZE[,VARRED])</td>
<td><strong>Beta distribution</strong> specified by two shape parameters.</td>
</tr>
<tr>
<td>cauchy(LOCATION,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Cauchy distribution</strong> specified by the location and the scale parameter.</td>
</tr>
<tr>
<td>chisquare(DF,SAMSIZE[,VARRED])</td>
<td><strong>Chi-Squared</strong> distribution specified by degrees of freedom.</td>
</tr>
<tr>
<td>exponential(RATE,SAMSIZE[,VARRED])</td>
<td><strong>Exponential distribution</strong> specified by rate of change.</td>
</tr>
<tr>
<td>f(DF_1,DF_2,SAMSIZE[,VARRED])</td>
<td><strong>F distribution</strong> specified by degrees of freedom. Note that the function sampleLSf uses another version of the F distribution than the function dF from the Stochastic Library.</td>
</tr>
<tr>
<td>gamma(SHAPE,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Gamma distribution</strong> specified by shape and scale parameter. Note that the function sampleLSgamma(A,B) is equivalent to the function dGamma(B,A) from the Stochastic Library.</td>
</tr>
<tr>
<td>gumbel(LOCATION,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Gumbel distribution</strong> specified by location and scale parameter.</td>
</tr>
<tr>
<td>laplace(LOCATION,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Laplace distribution</strong> specified by location and scale parameter.</td>
</tr>
<tr>
<td>logistic(LOCATION,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Logistic distribution</strong> specified by location and scale parameter.</td>
</tr>
<tr>
<td>lognormal(LOCATION,SCALE,SAMSIZE[,VARRED])</td>
<td><strong>Log Normal distribution</strong> specified by location and scale parameter.</td>
</tr>
<tr>
<td>normal(MEAN,STD_DEV,SAMSIZE[,VARRED])</td>
<td><strong>Normal distribution</strong> specified by given mean and standard deviation.</td>
</tr>
<tr>
<td>pareto(SCALE,SHAPE,SAMSIZE[,VARRED])</td>
<td><strong>Pareto distribution</strong> specified by shape and scale parameter.</td>
</tr>
<tr>
<td>studentt(DF,SAMSIZE[,VARRED])</td>
<td><strong>Student’s t-distribution</strong> specified by degrees of freedom.</td>
</tr>
<tr>
<td>triangular(LOW,MID,HIGH,SAMSIZE[,VARRED])</td>
<td><strong>Triangular distribution</strong> specified by lower and upper limit and mid value.</td>
</tr>
<tr>
<td>uniform(LOW,HIGH,SAMSIZE[,VARRED])</td>
<td><strong>Uniform distribution</strong> specified by the given bounds.</td>
</tr>
</tbody>
</table>
### Continuous Distribution

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>weibull(SCALE,SHAPE,SAMSIZE[,VARRED])</code></td>
<td><strong>Weibull distribution</strong> specified by scale and shape parameter. Note that the function <code>sampleLSweibull(A,B)</code> is equivalent to the function <code>dWeibull(B,A)</code> from the <em>Stochastic Library</em>.</td>
</tr>
</tbody>
</table>

### Discrete Distribution

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binomial(N,P,SAMSIZE[,VARRED])</code></td>
<td><strong>Binomial distribution</strong> specified by number of trials <code>N</code> and success probability <code>P</code> in each trial.</td>
</tr>
<tr>
<td><code>hypergeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])</code></td>
<td><strong>Hypergeometric distribution</strong> specified by total number of elements, number of good elements, and number of trials.</td>
</tr>
<tr>
<td><code>logarithmic(P-FACTOR,SAMSIZE[,VARRED])</code></td>
<td><strong>Logarithmic distribution</strong> specified by <code>P</code>-Factor. Note that the function <code>sampleLSlogarithmic</code> uses another version of the logarithmic distribution than <code>dLogarithmic</code> from the <em>Stochastic Library</em>.</td>
</tr>
<tr>
<td><code>negbinomial(SUCC,P,SAMSIZE[,VARRED])</code></td>
<td><strong>Negative Binomial distribution</strong> specified by the number of successes and the probability of success. The generated random number describes the number of failures until the defined number of successes is reached. Note that the function <code>sampleLSnegbinomial(R,P)</code> is equivalent to the function <code>dNegBinomial(R,P-1)</code> from the <em>Stochastic Library</em>.</td>
</tr>
<tr>
<td><code>poisson(MEAN,SAMSIZE[,VARRED])</code></td>
<td><strong>Poisson distribution</strong> specified by mean.</td>
</tr>
</tbody>
</table>

The following example illustrates the use of the sample generator and shows the effect of the functions `setCorrelation` and `induceCorrelation`:

```
$funcLibIn lsalib lsadclib

Functions normalSample / lsalib.SampleLSnormal /
    getSampleVal / lsalib.getSampleValues /
    setCor / lsalib.setCorrelation /
    indCor / lsalib.induceCorrelation /;

Scalars d "dummy"
    h "handle for first sample"
    k "handle for second sample" ;

Set i "sample index" / i01*i12 /;

Parameters sv_h(i) "sample values for handle h"
    sv_k(i) "sample values for handle k";

* generate two handles for 12 samples from normal distribution with mean 5 and std.dev. 2 each
  h = normalSample(5,2,12);
  k = normalSample(5,2,12);

* retrieve sample values from Lindo library
  loop(i, sv_h(i) = getSampleVal(h) );
  loop(i, sv_k(i) = getSampleVal(k) );
```
display sv_h, sv_k;

* set and induce a correlation between samples h and k
  d = setCor(h,k,-1);
  d = indCor(1);

* retrieve sample values again from correlated distribution
  loop(i, sv_h(i) = getSampleVal(h) );
  loop(i, sv_k(i) = getSampleVal(k) );

display sv_h, sv_k;

The resulting output shows that the values of sv_k are reordered according to the desired correlation:

---- 25 PARAMETER sv_h sample values for handle h
  i01 2.079,  i02 6.454,  i03 4.437,  i04 2.747,  i05 5.339,  i06 4.059,  i07 6.311,  i08 7.512,  i09 8.280,  i10 3.380,  i11 4.596,  i12 5.752

---- 25 PARAMETER sv_k sample values for handle k
  i01 5.509,  i02 3.021,  i03 7.550,  i04 6.002,  i05 4.227,  i06 0.704,  i07 3.890,  i08 9.474,  i09 5.084,  i10 4.592,  i11 3.311,  i12 6.442

---- 35 PARAMETER sv_h sample values for handle h
  i01 2.079,  i02 6.454,  i03 4.437,  i04 2.747,  i05 5.339,  i06 4.059,  i07 6.311,  i08 7.512,  i09 8.280,  i10 3.380,  i11 4.596,  i12 5.752

---- 35 PARAMETER sv_k sample values for handle k
  i01 7.550,  i02 3.021,  i03 9.474,  i04 6.442,  i05 4.592,  i06 0.704,  i07 3.890,  i08 6.002,  i09 3.311,  i10 5.084,  i11 4.227,  i12 5.509

4.47.3.5  The Mutex Library

The Mutex Library allows us to work with a process-level mutex - a program object that can be used to synchronize multiple processes. For example, a mutex can be used to coordinate mutually exclusive access to a resource shared between processes and, thus, prevent multiple processes from accessing the shared resource at the same time. A mutex object is created with a unique name and can be acquired or locked by only one process at a time. Each process waits to acquire the lock for (i.e. ownership of) the mutex object before executing the code that accesses the shared resource. After accessing the shared resource, the process releases or unlocks the mutex object and other processes waiting for the mutex can acquire the lock and continue execution.

The Mutex Library may be used to prevent simultaneous access to shared resources when running GAMS programs concurrently and the errors and hard-to-explain behavior that result. Such behavior is not unusual or surprising: GAMS even has some language features to spawn programs asynchronously (e.g. Asynchronous Execution), and these concurrent GAMS programs often share files. Consider the following example where we have two GAMS programs A and B concurrently running a common code that writes a GDX file containing a scalar x:
scalar x;
repeat
  execute_load 'x.gdx', x;
  x = x + 1;
  execute_unload 'x.gdx', x;
until x > 100;

To ensure correct execution of the code, we want to ensure that programs A and B have mutually exclusive access to the GDX file. In addition, we want the load-increment-unload sequence to occur atomically (i.e. no other process reads or writes the GDX file during this sequence). The extrinsic function library mtxcc provides us with some functions to accomplish this and similar tasks:

- **Create(x)**: Creates a mutex with id x in the system. Returns 0 on success and UNDF in case of error.
- **Lock(x)**: Request to acquire or lock a mutex with id x: it returns immediately if the mutex is available, otherwise it waits until the mutex becomes available. Returns 0 on success and UNDF in case of error. Any process waiting in this call will wait forever if the mutex is never unlocked.
- **Unlock(x)**: Unlocks or releases a previously locked mutex with id x. Returns 0 on success and UNDF in case of error. If processes are waiting to acquire a lock on x, one of them will acquire the lock. Note that the lock should be held by the unlocking process: behavior is undefined otherwise.
- **TryLock(x)**: Like Lock, but if the mutex is not available, return 1 immediately instead of waiting.
- **TimedLock(x,y)**: Like Lock, but instead of waiting forever, return 1 after waiting unsuccessfully for y milliseconds.
- **Delete(x)**: Erases a mutex with id x from the system. Returns 0 on success and UNDF in case of error. Note that a process does not need to own the mutex to erase it. If a mutex is deleted and there are remaining processes queued up waiting for the unlock, these processes will wait forever.

Continuing our previous example, we need to make the functions Lock and Unlock available so we can use them to protect the entire loop body by enclosing it in Lock/Unlock calls.

```plaintext
$funcLibIn mtxlib mtxcclib
Function lock / mtxlib.Lock /
  unlock / mtxlib.Unlock /;
scalar x;
repeat
  abort$lock(12345) 'Problems locking mutex';
  execute_load 'x.gdx', x;
  x = x + 1;
  execute_unload 'x.gdx', x;
  abort$unlock(12345) 'Problems unlocking mutex';
until x > 100;
```

Before we can use the mutex, we first have to create it. Also if both programs are done, we need to make sure that the named mutex is erased at the end of the programs, otherwise named mutexes collect in your system and consume resources (memory or disk depending on the implementation). Because the deletion of a named mutex might be forgotten or a program exits while holding the lock to the mutex or another concurrent system uses the same id, it is a good idea to find a unique mutex name, create it once, and erase it at the end as shown in the following complete example:
* generate a unique mutex ID
$eval mtxID round(frac(jnow)*24*60*60*1000)

$onEcho > x.gms
$funcLibIn mtxlib mtxcclib
Function lock / mtxlib.Lock /
   unlock / mtxlib.Unlock /;
scalar x;
repeat
   abort$lock(%mtxID%) 'Problems locking mutex';
   execute_load 'x.gdx', x;
   x = x + 1;
   put_utility 'log' / 'Program %prgID% increased x to ' x:0:0;
   execute_unlock 'x.gdx', x;
   abort$unlock(%mtxID%) 'Problems unlocking mutex';
until x >= 100;
$offEcho
* Initialize x.gdx
scalar x /1/;
$gdxOut x.gdx
$unload x
$gdxOut

$onEcho > create.gms
$funcLibIn mtxlib mtxcclib
Function create / mtxlib.Create /
abort$create(%mtxID%) 'problems creating mutex';
$offEcho
* Create the mutex
$call.checkErrorLevel gams create.gms lo=2

* Asynchronously spawn A and B
$call.async 'gams x.gms --prgID=A fileStem=A'
$eval jhA JobHandle
$call.async 'gams x.gms --prgID=B fileStem=B'
$eval jhB JobHandle

* Wait until both jobs are back
$set statA 0
$set statB 0
$label l1
$eval x sleep(1)
$if not %statA% == 2 $eval statA JobStatus(%jhA%)
$if not %statB% == 2 $eval statB JobStatus(%jhB%)
$if not %statA%%statB% == 22 $goto l1

$onEcho > erase.gms
$funcLibIn mtxlib mtxcclib
Function delete / mtxlib.Delete /;
abort$delete(%mtxID%) 'problems erasing mutex';
$offEcho
* Erase the mutex
$call.checkErrorLevel gams erase.gms lo=2

The log indicates which program updated x and the GDX file x.gdx at what stage:

Program B increased x to 20
Program B increased $x$ to 21
Program A increased $x$ to 22
Program A increased $x$ to 23

Models dicegrid and asyncincbi show more realistic uses for the mtxcc library.

### 4.47.4 Build Your Own Library

This section discusses the creation of a custom extrinsic function library. Before attempting to implement such a library, we suggest to study the example libraries for which source code and test models are available. These libraries are studied below.

**Attention**

Building extrinsic function libraries requires the knowledge of a regular programming language (like C/C++, Fortran, ...) and experience with handling compilers and linkers to build dynamically linked libraries. In some situation, it may be easier to use the simpler GAMS macros or batinclude files to define own functions.

**Note**

Extrinsic functions are limited to 20 scalar arguments and return a scalar value.

An extrinsic function library consists of a specification part and a number of callbacks to evaluate the defined functions at an input point.

The specification part is implemented by a callback `querylibrary`. It returns information about the library itself, available functions, their arguments, endogenous classification, etc. to the GAMS execution system. C, Delphi, or Fortran source code for this callback can be generated automatically by using the Python helper script `ql.py`. The script processes a specification file `*.spec`, which is specified as first argument. The format of this file is documented in the file `tri.spec`. Both `ql.py` and `tri.spec` are contained in the source of the trigonometric library examples in the GAMS test library, obtainable via:

```
$ testlib trilib01
$ unzip trisource.zip
```

If an extrinsic function will be used within equations of a GAMS model, next to the function value evaluation callback, also callbacks that compute first and second derivatives with respect to all endogenous arguments at an input point should be provided. Occasionally, this can be inconvenient. Observe that GAMS can use the function values at points close to the input point to estimate the derivative values using finite differences. However, this method is not as accurate as analytic derivatives and requires a number of function evaluations, thus the convenience comes at a price. The attribute `MaxDerivative` in the specification of a function signals GAMS the highest derivatives this function will provide. For higher order derivatives, GAMS will use finite differences to approximate the derivative values. However, a better alternative is often the use of automatic differentiation when implementing the function evaluation. This is demonstrated in the CPP Library Example, see section Automatic Differentiation for more details.

GAMS offers some support to check the implementation of of derivatives for extrinsic functions via the function suffixes `grad`, `gradn`, `hess` and `hessn`. These function suffixes are defined for intrinsic and extrinsic functions. For example, for an extrinsic function `userfunc`, the gradient evaluation that the user implemented may be called with `userfunc.grad`. Further, an approximation of the gradient by finite differences is available by calling `userfunc.gradn`. Comparing the results of these two calls can often help to check the implementation of the gradient. The same principle applies for the Hessian and the function suffixes `.hess` and `.hessn`. The GAMS options `FDDelta` and `FDOpt` can be used to influence the finite difference calculations. For more details, see model [DERIVTST] in the GAMS Model Library.
4.47.4.1 Example: Trigonometric Library

The Trigonometric Library serves as an example of how to code and build an extrinsic function library. The library is included in the GAMS distribution in binary form. In addition, the source code in C, Delphi, and Fortran is available in the include files of the models [TRILIB01], [TRILIB02] and [TRILIB03] respectively. The library implements the following extrinsic functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>setMode(MODE)</td>
<td>Sets mode globally. Possible values are 0 for radian and 1 for degree. May be overwritten by the optional argument MODE in the functions cosine and sine.</td>
<td>none</td>
</tr>
<tr>
<td>cosine(x[,MODE])</td>
<td>Returns the cosine of the argument x. Note that the argument MODE is optional, default setting: MODE = 0.</td>
<td>NLP</td>
</tr>
<tr>
<td>sine(x[,MODE])</td>
<td>Returns the sine of the argument x. Note that the argument MODE is optional, default setting: MODE = 0.</td>
<td>NLP</td>
</tr>
<tr>
<td>pi</td>
<td>Value of π = 3.141593...</td>
<td>any</td>
</tr>
</tbody>
</table>

The C implementation of this extrinsic function library can be found in the files tricclib.c and tricclibql.c. Together with the API specification file extrfunc.h, these files document the callbacks that need to be implemented by a GAMS extrinsic function library. The file tricclibql.c implements the querylibrary callback, which provides information about the library itself and the extrinsic functions it implements. For example, the information that the function cosine has an endogenous required first argument and an exogenous optional second argument is available from the querylibrary callback. The file tricclibql.c (and also the Delphi and Fortran90 equivalents tridclibql.inc and triifortlibql.f90) has been generated by the script ql.py by processing the specification file tri.spec.

4.47.4.2 Example: The CPP Library

The CPP library serves both as an example of how to use C++ to obtain gradients and Hessians "for free" and as a source of functions based on the multivariate Normal distribution. The library is available in compiled form and as C++ source. Test Library model [CPPLIB00] exercises the process of building a shared library from C++ source and doing some basic tests, while models [CPPLIB01], [CPPLIB02], [CPPLIB03], [CPPLIB04], and [CPPLIB05] are more thorough tests for the CPP library extrinsics shipped with the distribution. These functions are listed and described in the following table. Note that in keeping with the language conventions of statistics, PDF is shorthand for "probability density function" and CDF is shorthand for "cumulative distribution function".

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>End. Classif.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdfUVN(x)</td>
<td>PDF of uni-variate Normal distribution, see MathWorld or R</td>
<td>NLP</td>
</tr>
<tr>
<td>cdfUVN(x)</td>
<td>CDF of uni-variate Normal distribution, see MathWorld or R</td>
<td>NLP</td>
</tr>
<tr>
<td>pdfBVN(x,y,r)</td>
<td>PDF of bivariate Normal distribution, see MathWorld or R</td>
<td>NLP</td>
</tr>
<tr>
<td>cdfBVN(x,y,r)</td>
<td>CDF of bivariate Normal distribution, see MathWorld or R</td>
<td>NLP</td>
</tr>
<tr>
<td>pdfTVN(x,y,z,r21,r31,r32)</td>
<td>PDF of trivariate Normal distribution, see MathWorld or R</td>
<td>NLP</td>
</tr>
</tbody>
</table>

4.47.4.2.1 Automatic Differentiation Often, extrinsic functions are created in order to be used with endogenous arguments. In such cases it is necessary to provide first and second derivatives with
4.47 Extrinsic Functions

respect to these arguments in addition to the function values themselves. One way to compute these derivatives is via automatic differentiation techniques (see the article in Wikipedia for details).

Note that with C++ it is possible to overload the usual arithmetic operators (assignment, addition, multiplication, etc.) so that automatic differentiation occurs with little or no change to the function-only source code. This is the technique used to compute the derivatives in the CPP Library. Observe that the Test Library model [CPPLIB00] includes all the source code for the CPP Library and illustrates the steps needed to build the library from this source. The source is a working self-documentation of how the process of automatic differentiation works.

4.47.4.2.2 Multi-Variate Normal Distributions As shown in the table above, the CPP Library implements the PDF and CDF for the univariate, bivariate and trivariate standard Normal distributions. We use the standard Normal (mean of 0, standard deviation of 1) since intrinsic functions are limited to 20 arguments. The functions for the univariate case are included as convenient examples and should give results (nearly) identical to the functions pdfNormal and cdfNormal from the stochastic library. For the multivariate cases, the implementation is based on TVPACK from Alan Genz, with some modifications to allow for proper computation of derivatives. Note that we chose to implement the functions taking correlation coefficients as arguments, not a covariance matrix. The conversion from a covariance matrix to correlation coefficients is straightforward. The following R code describes this conversion. Here the package mnormt is used that computes the multivariate CDF with similar code to Genz:

```r
# start with a mean mu and variance-covariance matrix S1
x <- c(1.0, 3.0, 0.3)  
mu <- c(0.0, 1.0, -1.0)  
S1 <- matrix(c(0.0, 1.0, 1.5, 1.5, 1.0, 0.5, 1.5, 1.5, 9.0), nrow=3, ncol=3)  
v1 <- pmnorm(x=x, mean=mu, varcov=S1)  
  # convert to std normal with 3 correlation coeffs
R <- cov2cor(S1)  
sd <- sqrt(diag(S1))  
xn <- (x-mu) / sd  
v2 <- pmnorm(x=xn, mean=0, varcov=R)
```

Note that for the bivariate case there is one correlation coefficient \( r \). The CDF implementation is not quite accurate to machine precision, it has 14 or 15 digits of accuracy. The trivariate case includes 3 correlation coefficients, the 3 off-diagonal elements from the lower trangle of \( R \) above. The accuracy of the CDF depends on the inputs: it is higher when the correlation is nearly zero and lower as the condition number of \( R \) increases. Typically, an accuracy of \( 10^{-11} \) is all that can be achieved. In both multivariate cases, we recommend to avoid evaluating at or near points where the correlation matrix is degenerate. At nearly degenerate points, the accuracy of the distribution and density functions suffers. As the correlation matrix becomes degenerate, the distribution becomes degenerate too.

4.47.4.3 Remark: Stateful Libraries

While GAMS intrinsic functions are stateless, users may implement stateful extrinsic functions, i.e., functions that have some memory. There are two ways to achieve stateful extrinsic functions:

1. Library initialization (LibInit): at initialization time, the function library reads some data that is required to evaluate the provided functions. An example is the Piecewise Polynomial Library.

2. Previous function calls: function calls that alter the execution of successive function calls. An example is the function SetMode from the Trigonometric Library.

Attention

Altering the state of an extrinsic function library by function evaluation calls is problematic, since different parts of the GAMS system potentially use different instances of the function library.

For example, consider that the function SetMode of the Trigonometric Library is called via SetMode(1) before a solve statement. Unless option solvelink is set to 5, the solver will run in a separate process with a new instance of the function library and therefore will use the default mode, which is zero. Further, if solvelink is set to zero, the GAMS process will terminate in order to execute the solve statement and will restart a new GAMS process after the solve. The restarted GAMS process will load a fresh instance of the extrinsic function library, which has no memory of the value of mode from before the solve statement. This problem is demonstrated in the GAMS Test Library model [TRILIB04].

---
4.47.5 Extrinsic Functions vs. External Equations

In addition to extrinsic functions, GAMS offers another facility to include additional mathematical functions in GAMS: external equations. These equations are denoted by `equation type=x=`. A feasible solution for a model instance must satisfy all internal and external equations. External equations are introduced and discussed in chapter External Equations.

Similar to extrinsic functions, it is the users responsibility to provide routines that evaluates the external equation. Further, both facilities are especially pertinent to nonlinear models.

An overview of some characteristics of extrinsic functions and external equations is given in the following table:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Extrinsic Function</th>
<th>External Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of arguments</td>
<td>20</td>
<td>No limit</td>
</tr>
<tr>
<td>Available in statements</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Debugging support</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Returns Hessian to solver</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Consider the following example, which demonstrates how an equation $z = \int_{x_1}^{x_2} x \, dx$ may be formulated using either an extrinsic function and an external equation:

* using an extrinsic function
  
  ```gams
  e1.. integralx(x1,x2) =e= z;
  ```

* using an external equation
  
  ```gams
  e1.. 1*x1 + 2*x2 + 3*z =x= 1;
  ```

Note that the function `integralx` is assumed to be a user-defined extrinsic function, probably implemented as $\text{integralx}(x_1, x_2) = \frac{1}{2}(x_2^2 - x_1^2)$.

Note further, that the integers 1 to 3 in the external equation are mapped to indices for variables that are used in external functions inside an external module. The right-hand side of the external equation denotes the external equation number. It is assumed that the user-defined external equation implements the function $(x_1, x_2, z) \mapsto \int_{x_1}^{x_2} x \, dx - z$.

As a simple example for the use of an extrinsic function in a statement, consider the following assignment, which sets the level of variable $y$ to the value of $(\int_0^2 x \, dx)^2$:

```gams
y.l = sqr(integralx(0,2));
```

4.48 External Equations

GAMS provides a number of built-in or intrinsic functions for use in equations. Still, the extremely diverse set of application areas in which GAMS is used can create demand for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS External Equations Facility provides one means for managing this trade-off, since it allows users to import functions from an external library to define equations in a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with external equations. This includes the class of deterministic global solvers, see column "Global" in this table, while, for example, stochastic global solvers can work with external equations.
Both external equations and extrinsic functions aim to provide possibilities to extend GAMS by user-provided mathematical functions. However, there are fundamental differences in the use and implementation of both. For most situations, extrinsic functions should be preferred over external equations. See also Extrinsic Functions vs. External Equations.

Building external equation libraries requires the knowledge of a regular programming language (like C/C++, FORTRAN, ...) and experience with handling compilers and linkers to build dynamically linked libraries.

The external equation interface is not intended as a way to bypass some of the very useful model checking done by GAMS for models that are solved with NLP solvers. External equations are still assumed to be continuous with accurate and smooth first derivatives. The continuity assumption implies that external equations must have very low noise levels, considerably below the feasibility tolerance used by the solver. The assumption about accurate derivatives implies that derivatives must be computed more accurately than can be done with standard finite differences. If these assumptions are not satisfied, then there is no guarantee that the NLP solver can find a solution that has the mathematical properties of a local optimum, i.e., a solution that satisfies the Karush-Kuhn-Tucker conditions within the standard tolerances used by the solver.

In the following, connecting code written in FORTRAN, C, Delphi, or some other programming language to equations and variables in a GAMS model is described. These GAMS equations will be referred to as external equations and the compiled version of the programming routines will be referred to as the external module that defines the external functions. The form of the external module depends on the operating system that is used. The external module under Windows is a Dynamic Link Library (.dll) and the external module under Unix is a shared object (.so or .dylib). In principle, any language or system may be used to build the DLL or shared object that defines the external module, as long as the interface conventions are not changed.

The GAMS Test Library provides examples of external equations consisting of GAMS models and C, Delphi, Java, and FORTRAN code. For more details, see Section Examples in the GAMS Test Library.

The basic mechanism of external equations is to declare all the equations and variables using the usual GAMS syntax. The interpretation of the external equations is done in a special way. Instead of the usual semantic content, the external equations specify the mapping between the equation and variable names used in GAMS and the function and variable indices used in the external module. This mapping is described in Section Model Interface. The external module may be written in C, FORTRAN, or most other programming languages. Section Programming Interface describes the general definitions for an external module for C, Delphi, and FORTRAN from a programming language perspective. Note that the way the program is compiled and converted into an external module is system and compiler specific. The following Section Implementation gives detailed advice on various aspects of the implementation of the external module.

4.48.1 Examples in the GAMS Test Library

Model [TESTEXEQ] gives an overview of all examples in the GAMS Test Library and may be used to compile and run them. Note that the remainder of this chapter will reference examples that are listed in this model. Further, model [COMPLINK] may be used as a script to compile and link external equation libraries. Note that these models hardcore the path to the Java compiler and libraries and these paths will need to be adapted by the user when running the Java examples.

Observe that regardless of how external libraries are built, the examples (e.g. [EX1]) will by default solve a model without using external equations. To solve the example models with all kinds of different external equation libraries, they may be run with the argument...
Alternatively, only selected libraries may be used by using one or more of the following command line parameters:

--runall=1
--runC=1
--runC_cb=1
--runD=1
--runD_cb=1
--runF=1
--runF_cb=1
--runJ=1

4.48.2 Model Interface

4.48.2.1 External Equation Syntax

External equations that are used to specify the interface to the external module are declared in GAMS like any other equation. The syntax for the external equation definition statement is as follows:

```
eqn_name(index_list)[logical_condition(s)].. expression =x= expression ;
```

Note that the only difference to the usual equation definition is the use of the equation type =x=.

The equations defined by an external module are always interpreted as equality constraints with zero right-hand sides. Thus inequalities have to be converted to equalities by adding explicit slack variables, which will serve as additional external variables. A nonzero right-hand side need to be taken care of in the external equation implementation.

4.48.2.2 Mapping of external equations and variables to indices

Some mappings must be specified to link an external module to a GAMS model. External equations are assumed to be defined in terms of indices $i = 1...m$. These indices must be mapped to GAMS equation names. Similarly, the variables used inside the external functions are assumed to be defined in terms of indices $j = 1...n$. These indices must be mapped to GAMS variable names. Finally, the name of the external module must be specified. Note that GAMS solvers are typically designed for large models and rely on sparsity. The last part of the specification of a set of external equations is therefore the sparsity pattern of the external equations, i.e., which variables appear in which equations.

The value of the constant term of the external equation must be an integer, since the value of the constant maps the row of the GAMS equation to the index (in $1...m$) of the external equation. Several blocks of GAMS equations may be mapped to external equations using the =x= notation. The mapping between GAMS equations of type =x= and indices $1...m$ must be bijective (one-to-one). This means that two GAMS equations may not be mapped into the same external equation index and that there may not be any holes in the list of external equation indices. Although there may be any number of blocks of GAMS external equations, they must all map into and be implemented by one single external module.

The variable part of each external equation defines both the sparsity pattern of the external equation and the mapping from GAMS variables to the indices of the external variables. The variable part must be a sum of terms where each term is an integer times a variable. The existence of the term indicates that the variable involved is used in the external equation and that there is a corresponding derivative. The value of the coefficient defines the index of the external variable (in $1...n$) that the GAMS variable is mapped
to. For example, the term $5 \times Y$ indicates that the external equation depends on the GAMS variable $Y$ and that $Y$ is mapped to the 5th element in the vector of external variables. Clearly, if a variable appears in more than one external equation, then the value of its coefficient must be the same in each case.

Note that several blocks of GAMS variables may be used in external equations. In contrast to equations, where all rows in an equation block are either external or not, some columns in a variable block may be external while others are not. The mapping between GAMS variables that appear in equations of type $=x=$ and external variable indices $1...n$ must be bijective (one-to-one). This means that two GAMS columns may not be mapped into the same external variable index and that there may not be any holes in the list of external variable indices. Although there may be any number of blocks of GAMS variables mapped to external variables, they must all map into one single vector passed to the subroutine in the external module.

Observe that while some GAMS variables are external, there is no syntax provided to mark them as external variables. They may be used in non-external GAMS equations as well as external equations. Indeed, without this capability the model would be separable and the external equations and the functions they map to would be of little use.

Note

- As the coefficients and right-hand sides in the GAMS definition of external equations are interpreted as indices, users are not allowed to scale external equations and variables.
- External equations are treated in a special way, therefore the command line parameter and model attribute HoldFixed will not treat any fixed external variables as constants.

4.48.2.3 Name of external module

The name of the external module in which the external equations are implemented may be defined in a number of ways. By default, the external module is assumed to have the same name as the GAMS model with an extension that is operating system dependent. The extension is .dll for Windows, .dylib for MacOS X, and .so for any other Unix.

A custom name for the external module may be specified with a file statement. In this case the file name has to be listed as an additional item in the model statement. If the library extension is omitted in the file statement, GAMS will add the system-dependent extension automatically. This helps to make the model portable between different operating systems.

Consider the following simple example:

```plaintext
File myextfile / extern /;
Model mymodel / all, myextfile /;
```

When model `mymodel` is solved, GAMS will try to load the an external module file named `extern.so`, `extern.dylib`, or `extern.dll`, depending on the current operating system.

By default, the external module is assumed to be located in the directory external_equations in the GAMS standard locations or in the directory from which GAMS is called. A different location may be specified with an added path in the file statement.
### 4.48.3 Programming Interface

This section discusses C, Delphi, and FORTRAN interfaces to the GAMS external equations facility.

The external equation module need to provide a function called `GEFUNC`. The beginning of the external equation module typically looks as follows:

**C:**
```c
#define GEEXPORT
#include "geheader.h"
GE_API int GE_CALLCONV gefunc(int* icntr, double* x, double* f, double* d, msgcb, t msgcb)
```

The header file `geheader.h` can be found in the `testlib_ml` subdirectory of the GAMS distribution. It defines `GE_API` and `GE_CALLCONV` and the signature of the function `gefunc`. `GE_API` is used to indicate to the compiler whether the function should be exported or imported. Due to defining `GE_EXPORT` before including `geheader.h`, `GE_API` is defined such that the function will be marked for export (`_declspec(dllexport)` on Windows and `_visibility_("default")` with GCC). Further, `GE_CALLCONV` indicates the calling convention that should be used on Windows. Currently, this is defined to be `_stdcall`. On other operating systems, it is empty.

**FORTRAN:**
```fortran
Integer Function gefunc (icntr, x, f, d, msgcb)
C Control Buffer:
   Integer icntr(*)
C Numerical Input and Output:
   Double Precision x(*), f, d(*)
C Message Callback Routine
   External msgcb
```

**Delphi:**
```delphi
uses geheader_d;
Function GeFunc(var Icntr: ticntr;
    var x: tarray;
    var F: double;
    var D: tarray;
    MsgFunc: tMsgCallBack): integer; stdcall;
```

The unit file `geheader_d.pas` can be found in the `testlib_ml` subdirectory of the GAMS distribution.

In the following, the arguments of `GEFUNC` are described in detail.

#### 4.48.3.1 Control vector `icntr`

The array `icntr` is a control vector that is used to pack and communicate control information between GAMS and the external module. Some helpful definitions to work with the `icntr` array are provided by the files `geheader.h` (C), `geheader_d.pas` (Delphi), and `gehelper.f90` (Fortran 90). The array elements are the following:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>icntr[I_Length]</code></td>
<td>Holds the length of array <code>icntr</code> in number of elements. This is provided by GAMS.</td>
</tr>
<tr>
<td><code>icntr[I_Neq]</code></td>
<td>Number of external equation rows seen in the GAMS model. This is provided by GAMS.</td>
</tr>
<tr>
<td><code>icntr[I_Nvar]</code></td>
<td>Number of external variables seen in the GAMS model. This is provided by GAMS.</td>
</tr>
<tr>
<td><code>icntr[I_Nz]</code></td>
<td>Number of nonzero derivatives or Jacobian elements seen in the GAMS model. This is provided by GAMS.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| icntr[I_Mode] | Current mode of operation. This is provided by GAMS. The following values are possible:  
|              | **DOINIT**: Initialize. This will be the first call of GEFUNC, where initializations needed by the external module may be performed.  
|              | **DOTERM**: Terminate. This will be the last call of GEFUNC, where cleanup tasks needed by the external module may be performed.  
|              | **DOEVAL**: Function evaluation. External equations should be evaluated.  
|              | **DOCONSTDERIV**: Constant Derivatives. Information about constant derivatives should be provided.  
|              | **DOHVPROD**: Hessian-Vector product. The product between the Hessian of an external equation and a vector should be computed. See  
|              | Second Derivatives: Hessian times Vector for details.  |
| icntr[I_Eqno] | Index of the external equation to be evaluated during this call to GEFUNC. This is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL) and is a number between 1 and icntr[I_Neq], inclusive. Note that the external equation interface allows to communicate information about only one function at a time.  |
| icntr[I_Dofunc] | Flag whether function value should be computed. icntr[I_Dofunc] is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL). If set to 1, then GEFUNC must return the numerical value of the function indexed by icntr[I_Eqno] in the scalar f.  |
| icntr[I_Dodrv] | Flag whether derivative should be computed. icntr[I_Dodrv] is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL). If set to 1, then GEFUNC must return the numerical values of the derivatives of the function indexed by icntr[I_Eqno] in the array d.  |
| icntr[I_Newpt] | Flag for new point. icntr[I_Newpt] is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL). If set to 1, then the point x may be different from the previous call of GEFUNC. If set to 0, then x will not have changed since the previous call.  |
| icntr[I_Debug] | If icntr[I_Debug] is set to a nonzero value by the external equation module, then the functions GEstat and GElog will write all strings to a file called debugext.txt and flush the buffer immediately after writing. The string debugger may be used when a shared object crashes before GAMS has had an opportunity to display the messages. In FORTRAN, the string debugger will use FORTRAN unit icntr[I_Debug]. For more details see Section Message Output.  |
| icntr[I_Getfil] | Flag to request the name of a special directory or file from GAMS. The following values are possible:  
|              | I_Scr: Scratch directory,  
|              | I_Wrk: Working directory,  
|              | I_Sys: GAMS system directory,  
|              | I_Cntr: Control file.  
|              | For more information, see Section Communicating Data to the External Module via Files.  |
| icntr[I_Smode] | Flag for string mode. This is provided by GAMS. For details see Section Communicating Data to the External Module via Files.  |
| icntr[I_ConstDeriv] | Number of constant derivatives. Specifying this number during initialization is optional. For details see Section Constant Derivatives below.  |
| icntr[I_HVProd] | Indicator for use of Hessian-Vector product for second order derivatives. This entry is optional. For details see Section Second Derivatives: Hessian times Vector below.  |

Observe that FORTRAN programmers will have to replace the square brackets [] with parentheses ().
4.48.3.2 Evaluation point x

Argument x is an array with icntr[I_Nvar] elements and is provided by GAMS if GEFUNC is called in function evaluation mode (icntr[I_Mode] = DOEVAL). The individual elements of x will always be between the variable bounds defined in the GAMS model. During initialization and termination calls, x is not defined and the external module must not reference x. C programmers should index this array starting at zero, i.e., the first external variable is referenced as x[0].

4.48.3.3 Function value f

If icntr[I_Mode] = DOEVAL and icntr[I_Dofunc] = 1, then the external module must return the value of the external equation icntr[I_Eqno] in the scalar f. During initialization and termination calls, f must not be referenced.

4.48.3.4 Derivative vector d

If icntr[I_Mode] = DOEVAL and icntr[I_Dodrv] = 1, then the external module must return the values of the derivatives of external function icntr[I_Eqno] with respect to all variables in the array d. The derivative with respect to variable x[i] is returned in d[i]. It is sufficient to set only those positions in d that correspond to variables actually appearing in equation icntr[I_Eqno]. Other positions are not being used by GAMS and may be left undefined. During initialization and termination calls, d must not be referenced.

4.48.3.5 Message callback msgcb

This argument is the address of a message callback routine that can be used to write messages to the status and/or log files of the GAMS process. Its type definition in C is as follows:

```c
typedef void (GENCALLCONV * msgcb_t) (const int* mode, const int* nchars, const char* buf, int len);
```

The argument mode is used to point to an integer which indicates where messages should be written to. This integer can be set to the following values:

- LOGFILE (1): Write the message to the log file only.
- STAFILE (2): Write the message to the status file only.
- LOGFILE | STAFILE (3): Write the message to both the log file and the status file. Observe that the symbol | denotes the bitwise logical OR in C.

The argument nchars points to an integer that specifies the number of bytes contained in the message (excluding the \0-terminator if there is one present). Thus, in C, nchars is typically set to strlen(buf). The argument buf is a pointer to the character array containing the message to be printed. Finally, len is the size or length of the string buf, thus it is typically the same as *nchars.

Calling the message callback msgcb from C is straightforward. Note that the arguments mode, nchars, and buf are all call-by-reference and that addresses, not values, must be used. However, the argument len is call-by-value and *nchars should be passed as its value.

If the implementation is done in Delphi or Visual Basic, observe that pointers of all types are 4-byte quantities on a 32bit system and 8-byte quantities on a 64bit system. Integers are 4 bytes.

Calling this routine from a FORTRAN environment is a bit more complicated due to the different ways that FORTRAN compilers handle strings. The Unix convention - at least the convention observed on all systems for which GAMS is built - is that strings are passed by reference. In addition, the length of the string is passed by value as a hidden 4-byte quantity appended to the end of the argument list. This is the reason for including len as the last argument in msgcb. The argument len facilitates making FORTRAN callbacks in a Unix environment like the following:

```fortran
character*(*) msgbuf
int nchars, charcount
nchars = charcount(msgbuf)
call MSGCB (mode, nchars, msgbuf)
```
4.48 External Equations

4.48.3.6 Return code

The function GEFUNC must return one of the following status codes:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error occurred.</td>
</tr>
<tr>
<td>1</td>
<td>A function evaluation error was encountered. GAMS should not use the content of f and/or d, but GEFUNC has recovered from the error and is ready to be called at a new point. This status code should only be used in function evaluation mode (icntr[1_Mode]=DOEVAL).</td>
</tr>
<tr>
<td>2</td>
<td>Fatal error. If this value is returned during the initialization call, then GAMS should abort immediately. It may be returned by GEFUNC during the initial call if some initializations did not work correctly, or if some of the size values in icntr had unexpected values. It may also be returned during function evaluation mode (icntr[1_Mode]=DOEVAL) if the external module has experienced problems from which it cannot recover.</td>
</tr>
</tbody>
</table>

4.48.4 Implementation

After describing the function GEFUNC in Section Programming Interface above, this section offers some practical comments on implementing GEFUNC.

4.48.4.1 Compiling and Linking

The examples for GAMS external equations contain a set of GAMS models for compiling the code on various systems using various compilers. Note that the compiler and linker flags shown in these examples should be used to ensure that the modules conform to the interface standard. In addition, the appropriate include file (geheader.h, geheader_d.pas, gehelper.f90) should be used.

4.48.4.2 Initialization Mode

The initialization mode should always check whether the external equations have the expected size: icntr[1_Neq], icntr[1_Nvar] and icntr[1_Nz] have to be tested against fixed expected values or values derived from some external data set.

The initialization mode may be used for several purposes like allocating memory and initializing numerical information or mapping information needed by the function evaluations that will follow. Data can be computed or read from external data sources or it can be derived from calls to an external database. Note that data that is shared with GAMS may be written to a file from GAMS using the put statement and then read in GEFUNC. Note further, that users must close the put file with a putclose statement before the solve statement. Observe that memory used to hold information from one invocation of GEFUNC to the next should be static. For FORTRAN it should either be in a Common block or it should be included in a Save statement.

4.48.4.3 Termination Mode

The termination mode may be used to perform some clean-up tasks like computing statistics, closing files, and returning memory.
4.48.4.4 Evaluation Mode

The bulk of the computational work will usually be in evaluation mode. Observe that GEFUNC only works with one equation at a time. One of the reasons for this choice is that the addressing of derivatives becomes very simple: there is one derivative for each variable and they have the same index in d and x, respectively.

In some applications several functions are naturally evaluated together, for example, because all functions are computed in some joint integration routine. The icntr[I_Newpt] flag is included for these applications. Using this flag, an implementation could evaluate all functions using a common routine when icntr[I_Newpt] equals 1 and saves the function and derivative values. Additionally, it returns the values corresponding to equation icntr[I_Eqno]. In subsequent calls to GEFUNC, icntr[I_Newpt] will likely be zero and the function and derivative values can quickly be extracted from the previously computed (and saved) information.

4.48.4.5 Evaluation Errors

It is good modeling practice to add bounds to the variables in such a way that all nonlinear functions are defined for all values of the variables within the bounds. Most solvers will also guarantee that nonlinear functions are called only when all entries of the vector x are between the bounds. However, it may not be practical to add all the necessary bounds and the implementation of GEFUNC should therefore capture evaluation errors such as division by zero, taking the logarithm of non-positive numbers, overflow in exponentiation, etc. If an equation cannot be evaluated at the given point, function GEFUNC should simply let the solver know about this situation by returning the value 1. The solver may then be able to backtrack to a safe point and continue the optimization from there.

Attention

System-default or user-defined functions that handle evaluation errors (for example, the C library function matherr()) will sometimes not work in the same way inside a DLL or a shared object as they do in a self-contained program or a static library.

4.48.4.6 Message Output

External modules can send messages to the GAMS status file (usually the listing file) and the GAMS log file (usually the screen). Messages to be included in the GAMS status file can be buffered using the GEstat utility routine described below and messages to be included in the GAMS log file can be buffered using the GElog utility routine. Note that it is not possible to open these files for writing in the external module since GAMS or the solver process controls them.

Moreover, messages may be sent to both the status and log file without buffering, using the message callback msgcb. This removes the limit imposed by the size of the message buffer and may also make debugging somewhat simpler, since there is no need to worry about messages that never got flushed from the buffer. As it may be difficult or impossible to use the message callback from some environments, both the buffered and unbuffered techniques are provided.

Note that the two techniques for sending messages (buffered via GEstat and GElog and unbuffered via the message callback msgcb) are complementary. Either one or the other may be used, but if both are used in the same external module, the buffered messages will be printed after the unbuffered messages.
4.48.4.6.1 GEstat: The Utility Routine for Writing Messages to the Status File

GEstat is provided in the appropriate include file (Fortran 90: gehelper.f90, C: geheader.h, Delphi: geheader_d.pas). It is used to communicate messages that should be written to the GAMS status file. The function definition follows:

FORTRAN:

```fortran
subroutine gestat (icntr, line)
C Control Buffer:
    integer icntr(*)
C input parameters:
    character(*) line
C:
    void GEstat(int* icntr, char* line)
Delphi:
    Procedure GeStat(var icntr: ticntr; const s: shortstring);
```

Note that the first argument, `icntr`, must be passed through from the call of the function GEFUNC. The content of the argument `line` (or `s` in Delphi) is packed into the control buffer as one line. When GEFUNC returns, the content of the buffer will be written to the GAMS status file. GEstat may be called several times, each time with one line. Observe that `line` should not be longer than 132 characters and the overall amount of information written in one call to GEFUNC should not exceed 1000 characters. Further, `line` should not contain any special characters such as new-line or tab.

In practice, GEstat is often used with calls like the following:

FORTRAN:

```fortran
call GESTAT (icntr, ' ')
call GESTAT (icntr, '**** External module based on abc.for')
C:
GEstat (icntr, ' ')
GEstat (icntr, '**** External module based on abc.c')
Delphi:
gestat(icntr,' ');
gestat(icntr,'**** External module based on abc.dpr');
```

4.48.4.6.2 GElog: The Utility Routine for Writing Messages to the Log File

Like GEstat, GElog is provided in the appropriate include file. It is used to communicate messages that should be written to the GAMS log file. Note that by default, the log file is the screen. Alternatively, log may be written to a file that is specified with the GAMS command line parameter LogFile. The function definition of GElog follows:

FORTRAN:

```fortran
subroutine gelog ( Icntr, line )
C Control Buffer:
    integer Icntr(*)
C input parameters:
    character(*) line
C:
    void GElog(int* icntr, char* line)
Delphi:
    Procedure GeLog(var icntr: ticntr; const s: shortstring);
```

Note that GElog behaves exactly like GEstat, with the status file replaced by the log file. The content of `line` is written to a buffer that in turn is written to the log file when GEFUNC returns.

Observe that it is not possible to write directly to the screen with some combinations of operating system and compiler. This may also depend on the options or flags that are used to build the external module.

Attention

On some systems writing directly to the screen may cause the external module to crash. Therefore, it is advised not to write to the screen as a method for debugging, unless it is clear that it works. Otherwise the module may continue to crash because of the debugging statements after all other errors have been removed. Writing to a file and flushing the buffer is recommended as a safe alternative.
4.48.4.7 Communicating Data to the External Module via Files

Some external equations will need data from the GAMS program. This data may be passed on via one or more files written using put statements. Usually, such put files will be written in the current directory and the external module will look for them in the current directory. However, if users need to run multiple copies of the same model at the same time, data files should be written in the GAMS scratch directory and the external module should be directed to look for the data files in the scratch directory.

Note that a put file may be defined to be located in the scratch directory with the following file statement in the GAMS model:

File f / '%gams.scrdir%filename' /;

Observe that if the extension .dat is used, GAMS will remove the file from the scratch directory after the run. If another extension is used and the file is not deleted, GAMS will complain about an unexpected file when it cleans up after the run. The external module can receive the name of the scratch directory from GAMS during initialization by setting icntr[I_Getfil] to I_scr and returning immediately. GAMS will then store the name of the scratch directory and length of the name in the communication buffer and call GEFUNC in initialization mode again. Note that GEFUNC will now be called with the sub-mode icntr[I_Smode] set to I_Scr. Then the name may be extracted using the following FORTRAN call:

call GENAME(Icntr, Scrlen, Scrdir)

Here, Scrdir (declared as character(255)) will receive the scratch directory and Scrlen (declared as integer) will receive the actual length of Scrdir. In C, the call takes the following form:

char scratchDir[255];
int scratchDirLen;
scratchDirLen = GEname(icntr, scratchDir, sizeof(scratchDir));

Here the routine will return the number or characters transferred to the buffer scratchDir if successful and the value -1 otherwise. If there is space, a terminating '\0'-byte will be written to scratchDir. If the value returned is equal to sizeof(scratchDir), then the string returned will not be '\0'-terminated and may have been truncated as well.

Observe that it is possible to get other directory or file names by specifying other values in icntr[I_Getfil]. After setting this flag, GEFUNC must always return immediately.

For examples, see models [EX5] and [EXMCP3] and their respective FORTRAN and C source files.

4.48.4.8 Constant Derivatives

Some solvers, like the CONOPT solvers, can take advantage of the knowledge about constant derivatives in equations, which are a result of linear terms. This can be especially useful if an external equation represents an equation like \( Y = f(X) \), where \( Y \) is unbounded, since variable \( Y \) can then be substituted by \( f(X) \).

However, with the external module interface as described so far, the solver cannot know which variables appear linearly in the external equations. An optional extension allows to indicate that some of the relationships are linear. This can be activated by returning the number of constant derivatives in icntr[I_ConstDeriv] during the call of GEFUNC in initialization mode.

If the solver can use this information (not all solvers will), then GEFUNC will be called again repetitively with icntr[I_Mode] set to DOCONSTDERIV, once for each external equation, with its index specified as usual in icntr[I_Eqno]. For each of these calls, values of all constant derivatives must be specified in the array d. The remaining elements of d, both those corresponding to varying derivatives and to zeros, must be left untouched. These special calls will take place after the initialization call and before the first function evaluation call. Note that in these calls other flags like icntr[I_Dofunc] and icntr[I_Dodrv] and the array x will not be defined.

For an example, see model [EX4X] with the corresponding Fortran 90 and C source files ex4xf_cb.f90 and ex4xc_cb.c, respectively. It is instructive to compare these files to the corresponding files without constant derivatives: ex4f_cb.f90 and ex4c_cb.c.
4.48 External Equations

4.48.4.9 Second Derivatives: Hessian times Vector

External modules cannot provide a solver with the Hessian matrix of external equations. However, some solvers have particular options for internally approximating the Hessian. For example, see the hessian_approximation option for IPOPT or hessopt for KNITRO. Further, the solver CONOPT can take advantage of second order information in the form of the product Hessian matrix \( \nabla^2 f(x) \) times a vector \( v \). This special form can be used for external equations by setting \( \text{icntr}[\text{HVprod}] \) to 1 during the call of GEFUNC in initialization mode.

If the solver can use this information (not all solvers will), then GEFUNC may be called with \( \text{icntr}[\text{Mode}] \) set to DOHVPROD to request this operation. \( \text{icntr}[\text{Eqno}] \) will hold the equation number and the array \( x \) will hold the values of the variables \( (x) \) in its first \( \text{icntr}[\text{NVar}] \) positions and a vector \( v \) in the following \( \text{icntr}[\text{Nvar}] \) positions. GEFUNC should evaluate and return \( d = \nabla^2 f(x) v \) for the particular external equation \( f \) at the particular point \( x \). Note that \( d \) (which is otherwise used for the derivative vector) will have been initialized to zero by GAMS.

\( \nabla^2 f(x) v \) will often be needed for several vectors \( v \) at the same point \( x \). Therefore, \( \text{icntr}[\text{Newpt}] \) will be used to indicate changes in \( x \) in the usual way.

Note that model [EX1X] with the corresponding Fortran 90 source file shows how to use both constant and second derivatives.

4.48.4.10 Debugging

Implementing external equations brings a number of new potential error sources which GAMS cannot protect against as well as with pure GAMS models. For example, the argument lists in the C or FORTRAN code may be incorrect or the linking process may create an incorrect external module. There is little GAMS can do to help users with this type of errors. It is recommended to carefully follow the examples and output debug messages during the setup calls, for example using the utility routines GEstat and GElog.

Once the overall setup is correct and GAMS can establish proper communication with the external module, there may still be numerical errors where the function values and the derivatives do not match.

Note that the solver CONOPT will by default call its Function and Derivative Debugger in the initial point, if a model has any external equations. The debugger will check that the functions only depend on the variables that are defined in the sparsity pattern and that derivatives computed by numerical perturbation are consistent with the derivatives computed by the external module. If an error is found, CONOPT will stop immediately with an appropriate message. For examples, see the GAMS Test Library models [ER1], [ER2], and [ER3], which illustrate different types of errors. The respective error messages will appear if CONOPT is used as the NLP solver. Note that comments about the errors may be found in the C or FORTRAN source code.

Observe that several types of errors cannot be detected. Derivatives that are computed in the external module and are returned in positions that were not defined in the sparsity pattern in GAMS will be filtered out by the interface and will therefore not be detected. Similarly, derivatives that should be computed but are forgotten, may inherit values from the same derivatives in another equation computed earlier. Finally, fixed variables cannot be perturbed, thus errors related to these variables will usually not be detected.
4.49 GAMS Return Codes

The most convenient way to embed a GAMS program in a different program (e.g. C#, Java, Python, ...) is the object orient API to GAMS. A much simpler but less powerful way is to make a call to the GAMS executable (gams.exe (Windows) or gams (Unix)) with appropriate parameters from your program. Different languages and operating systems have different ways of accomplishing such a task. A common way to communicate a small piece of information from the GAMS program to the caller program is the exit status or return code (see e.g. https://en.wikipedia.org/wiki/Exit_status). GAMS return codes allow the caller to get some information about the status of the finished GAMS job. Note that return codes do not provide information about a model inside the GAMS job: the model may have been infeasible or may have failed in another way while the return code says all is fine. In fact, there may be multiple solves in a GAMS job, so even conceptually it is not possible to return solution status codes in the return code. The user cannot explicitly set the return code but can trigger action (e.g. abort; stop;) that result in a specific return code (here execution error (3)).

Note
In general, the value of zero for return codes denotes success and non-zero values denote failure.

We first demonstrate how to access return codes from within GAMS with a self-explanatory example and then list all return codes. GAMS has the ability to call other programs via the $call (compile time) and execute (execution time) commands. Naturally, GAMS can call GAMS in a recursive fashion. There are different methods to access the return code of such a sub-GAMS job:

```gams
$onecho > x.gms
set a /1,2,3/;
$offecho
* Compile time access inside to return values
* Example for non-zero return code
$call gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%
* Check the return code via if [not] errorlevel n
$if not errorlevel 1 $abort expect a errorlevel >= 1
* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%
* Example for zero return code
$call gams x.gms lo=0 > %system.nullfile%
* Check the return code via if [not] errorlevel n
$if errorlevel 1 $abort expect a errorlevel <= 0
* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%
* Runtime
* Example for non-zero return code
execute 'gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%';
* Access the return code as function value of function errorLevel
```


scalar myerrorlevel;
myerrorlevel = errorlevel;
display 'should be 6:', myerrorlevel;

* Example for zero return code
execute 'gams x.gms lo=0 > %system.nullfile%';

* Access the return code as function value of function errorLevel
myerrorlevel = errorlevel;
display 'should be 0:', myerrorlevel;

Command line interpreters or shells are a powerful way for job control and can naturally also run GAMS jobs. Here are two examples that demonstrate how to access (actually echo) the return code from GAMS in such environments:

Unix shell (e.g. bash):

```
$ gams mymodel ... 
$ echo $? 
```

Here $? is the environment variable that stores the return code from the last run.

Windows (cmd.exe):

```
C:\tmp> gams mymodel ... 
C:\tmp> echo %errorlevel%
```

Here %errorlevel% is the environment variable that stores the return code from the last run.

Note

On UNIX, return codes are treated modulo 256, so the return code 400 will be 144 on Unix. The return code modulo 256 is given in parenthesis in the table, if different from the return code.

### 4.49.1 List of the Error/Return Codes

The following table gives the list of the GAMS return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal return</td>
</tr>
<tr>
<td>1</td>
<td>Solver is to be called, the system should never return this number</td>
</tr>
<tr>
<td>2</td>
<td>There was a compilation error</td>
</tr>
<tr>
<td>3</td>
<td>There was an execution error</td>
</tr>
<tr>
<td>4</td>
<td>System limits were reached</td>
</tr>
<tr>
<td>5</td>
<td>There was a file error</td>
</tr>
<tr>
<td>6</td>
<td>There was a parameter error</td>
</tr>
<tr>
<td>7</td>
<td>There was a licensing error</td>
</tr>
<tr>
<td>8</td>
<td>There was a GAMS system error</td>
</tr>
<tr>
<td>9</td>
<td>GAMS could not be started</td>
</tr>
<tr>
<td>10</td>
<td>Out of memory</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>11</td>
<td>Out of disk</td>
</tr>
<tr>
<td>109</td>
<td>Could not create process/scratch directory</td>
</tr>
<tr>
<td>110</td>
<td>Too many process/scratch directories</td>
</tr>
<tr>
<td>112</td>
<td>Could not delete the process/scratch directory</td>
</tr>
<tr>
<td>113</td>
<td>Could not write the script <strong>gamsnext</strong></td>
</tr>
<tr>
<td>114</td>
<td>Could not write the <strong>parameter</strong> file</td>
</tr>
<tr>
<td>115</td>
<td>Could not read environment variable</td>
</tr>
<tr>
<td>400 (144)</td>
<td>Could not spawn the GAMS language compiler (<strong>gamscmex</strong>)</td>
</tr>
<tr>
<td>401 (145)</td>
<td>Current directory (<strong>curdir</strong>) does not exist</td>
</tr>
<tr>
<td>402 (146)</td>
<td>Cannot set current directory (<strong>curdir</strong>)</td>
</tr>
<tr>
<td>404 (148)</td>
<td>Blank in system directory (UNIX only)</td>
</tr>
<tr>
<td>405 (149)</td>
<td>Blank in current directory (UNIX only)</td>
</tr>
<tr>
<td>406 (150)</td>
<td>Blank in scratch extension (<strong>scrext</strong>)</td>
</tr>
<tr>
<td>407 (151)</td>
<td>Unexpected <strong>cmexRC</strong></td>
</tr>
<tr>
<td>408 (152)</td>
<td>Could not find the process directory (<strong>procdir</strong>)</td>
</tr>
<tr>
<td>409 (153)</td>
<td>CMEX library not be found (experimental)</td>
</tr>
<tr>
<td>410 (154)</td>
<td>Entry point in CMEX library could not be found (experimental)</td>
</tr>
<tr>
<td>411 (155)</td>
<td>Blank in process directory (UNIX only)</td>
</tr>
<tr>
<td>412 (156)</td>
<td>Blank in scratch directory (UNIX only)</td>
</tr>
<tr>
<td>909 (141)</td>
<td>Cannot add path / unknown UNIX environment / cannot set environment variable</td>
</tr>
<tr>
<td>1000 (232)</td>
<td>Driver error: incorrect command line parameters for <strong>gams</strong></td>
</tr>
<tr>
<td>2000 (208)</td>
<td>Driver error: internal error: cannot install interrupt handler</td>
</tr>
<tr>
<td>3000 (184)</td>
<td>Driver error: problems getting current directory</td>
</tr>
<tr>
<td>4000 (160)</td>
<td>Driver error: internal error: GAMS compile and execute module not found</td>
</tr>
<tr>
<td>5000 (126)</td>
<td>Driver error: internal error: cannot load option handling library</td>
</tr>
</tbody>
</table>

Note that error 3000 is sometimes caused by specifying the current directory in Microsoft UNC format. The return codes smaller than 100 come from the GAMS compiler and execution system (**gamscmex**) while the codes above 100 come from the GAMS driver program (**gams**).

### 4.50 GAMS Data eXchange (GDX)

This document describes the **GDX** (GAMS Data eXchange) facilities available in GAMS. The GDX facilities provide basic functionalities for exchanging GAMS data such as read and write. In addition to these facilities, there are a number of **GDX Tools** for exchanging data between GAMS and other data sources as well as for performing specific operations on a GDX file.

A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements. Among other usages a GDX file can also be used to prepare data for a GAMS model, pass results of a GAMS model into different programs, and pass results into GAMS from different programs.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering. See also **General notes on GDX files**.
4.50.1 Reading a GDX file

Reading a GDX file into a GAMS model can be done during the compile phase or the execution phase.

4.50.1.1 Compile Phase

During compilation, we can use Dollar Control Options to read data from a GDX file. Reading during the compile phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. The following Dollar Control Options are available for reading data from a GDX file into GAMS during compilation of a GAMS model:

- `$gdxin`
- `$load`
- `$loaddc`
- `$loadm`, `$loaddr`, `$loaddcm`, and `$loaddcr`
- `$loadidx`

4.50.1.1.1 `$gdxin` The `$gdxin` command is used in a sequence either to load specified items from a GDX file or to close the specified GDX file. The statement with `$gdxin` can be used in one of the following forms:

$$
gdxin \text{ file\_name} \\
\text{gdxin}
$$

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; read from the current working directory)</td>
</tr>
<tr>
<td></td>
<td>(no file_name) closes the GDX file</td>
</tr>
</tbody>
</table>

See Example 1 - Reading a GDX File and Example 2 - Reading a GDX File on how to use `$gdxin`.

4.50.1.1.2 `$load` The `$load` command loads specified items from a GDX file. The statement containing `$load` is preceded and succeeded by the `$gdxin` statements where

- the preceding `$gdxin` specifies the GDX file name and opens the file.
- the succeeding `$gdxin` closes the GDX file.

The statement with `$load` command can be used in one of the following forms:
$load
$load id=*
$load id1 id2 ... idn
$load id1=gdxid1 id2=gdxid2
$load id1<=gdxid1 id2<=gdxid2.dim3

where:

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no identifiers)</td>
<td>Reads all symbols from the gdx file and writes to the listing file</td>
</tr>
<tr>
<td>id=*</td>
<td>Loads all unique elements from the gdx file into set id</td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Read GAMS symbols id1, id2, ... idn from the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file</td>
</tr>
<tr>
<td>id1&lt;gdxid1 id2&lt;gdxid2</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (&lt;). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File.</td>
</tr>
<tr>
<td>id1&lt;=gdxid1 id2&lt;=gdxid2</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (&lt;=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File.</td>
</tr>
</tbody>
</table>

Note

- One can load the level values of a variable into a parameter of the same dimension using the syntax parametername=var.l as follows

  parameter storexlevel(i,j);
  $gdxin tran2
  $load storexlevel=x.l
  $gdxin
  display storexlevel;

- Loading the level values for a variable into the same variable (x=x.l) works but loads everything including bounds, scales, marginals and levels.

See Example 1 - Reading a GDX File and Example 2 - Reading a GDX File on how to use $load.

4.50.1.1.3 $loaddc  The $loaddc command lists all symbols in the GDX file and will cause a compilation error when the data read causes a domain violation. In contrast the $load command simply ignores elements that are not in the domain. See also Compile Phase Reading Data using $load.

The statement with $loaddc can be used in one of the following forms:
$loaddc
$loaddc id=*  
$loaddc id1 id2 ... idn  
$loaddc id1=gdxid1 id2=gdxid2  
$loaddc id1<=gdxid1 id2<=gdxid2.dim3

where:

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no identifiers)</td>
<td>Reads all symbols from the gdx file and writes to the listing file</td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Read GAMS symbols id1, id2, … idn from the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file</td>
</tr>
<tr>
<td>id1&lt;gdxid1 id2&lt;gdxid2.dim3</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (&lt;). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File.</td>
</tr>
<tr>
<td>id1&lt;=gdxid1 id2&lt;=gdxid2.dim3</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (&lt;=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File.</td>
</tr>
</tbody>
</table>

See Example 4 - Reading a GDX File on how to use $loaddc.

4.50.1.1.4 $loadm, $loadr, $loaddcm, and $loaddcr  The $loadm, $loadr, $loaddcm, and $loaddcr are additional forms of the $load and $loaddc directives. The M indicates a merge and the R indicates a full replacement.

The statements with $loadm can be used in one of the following forms:

$loadm
$loadm id1 id2 ... idn  
$loadm id1=gdxid1 id2=gdxid2  
$loadm id1<gdxid1 id2<gdxid2.dim3  
$loadm id1<=gdxid1 id2<=gdxid2.dim3

The statement with $loadr can be used in one of the following forms:

$loadr
$loadr id1 id2 ... idn  
$loadr id1=gdxid1 id2=gdxid2  
$loadr id1<gdxid1 id2<gdxid2.dim3  
$loadr id1<=gdxid1 id2<=gdxid2.dim3
The statement with \$loaddcm can be used in one of the following forms:

\$loaddcm
\$loaddcm id1 id2 ... idn
\$loaddcm id1=gdxid1 id2=gdxid2
\$loaddcm id1<gdxid1 id2<gdxid2.dim3
\$loaddcm id1<=gdxid1 id2<=gdxid2.dim3

The statement with \$loaddcr can be used in one of the following forms:

\$loaddcr
\$loaddcr id1 id2 ... idn
\$loaddcr id1=gdxid1 id2=gdxid2
\$loaddcr id1<gdxid1 id2<gdxid2.dim3
\$loaddcr id1<=gdxid1 id2<=gdxid2.dim3

where:

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no identifiers)</td>
<td>Reads all symbols from the gdx file and writes to the listing file</td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Read GAMS symbols id1, id2, ..., idn from the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file</td>
</tr>
<tr>
<td>id1&lt;gdxid1 id2&lt;gdxid2.dim3</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (&lt;). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set.</td>
</tr>
<tr>
<td>id1&lt;=gdxid1 id2&lt;=gdxid2.dim3</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (&lt;=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set.</td>
</tr>
</tbody>
</table>

See Example 4 - Reading a GDX File and \[load8\] examples on how to use \$loadm and \$loaddcm.

4.50.1.1.5 \$loadidx The command \$loadidx read GAMS symbols from the GDX file. Each symbol should have been written using an indexed write.

The statement with \$loadidx can be used in one of the following forms:

\$loadidx
\$loadidx id1 id2 ... idn
\$loadidx id1=gdxid1 id2=gdxid2

where:
4.50 GAMS Data eXchange (GDX)

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no identifiers)</td>
<td>Reads all symbols from the gdx file and writes to the listing file</td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Read GAMS symbols id1 id2 ... idn from the GDX file. Each symbol should have been written using an indexed write; see also execute_unloadidx</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file. Each symbol should have been written using an indexed write; see also execute_unloadidx</td>
</tr>
</tbody>
</table>

See [ldidx01] example on how to use $loadidx.

4.50.1.2 Execution Phase

When reading data from a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to read data from a GDX file during execution phase are:

- execute_load
- execute_loaddc
- execute_loadpoint

4.50.1.2.1 execute_load The `execute_load` statement acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. The same restrictions apply as in an assignment statement: we cannot assign to a set that is used as a domain, or to a set used as a loop control.

Sets defining domains cannot be loaded. However sets that are subsets of existing sets and do not define new elements can be loaded at execution time (Domain defining sets can be loaded can at compile time using $load).

The statement with `execute_load` can be used in one of the following forms:

```plaintext
execute_load 'file_name', id1, id2, ..., idn ;
execute_load 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;
execute_load 'file_name', setid=* ;
```

where

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; read from the current working directory)</td>
</tr>
<tr>
<td>id1, id2, ..., idn</td>
<td>Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional</td>
</tr>
<tr>
<td>id1=gdxid1, id2=gdxid2, ..., id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional</td>
</tr>
</tbody>
</table>
| setid=* | Allow to load the universe of labels from a GDX file into a set. Note, that only labels
Note

- items must be declared with Set, Parameter, Scalar, Variable or Equation statements before
the execute_load appears.
- when loading data domain checking is not enforced so that when an item is resident in a GDX
file for set elements not present in the current file these items are ignored and do not create
errors or cause generation of any messages. The execute_loaddc variant checks to see that the
domains match.

See [load11] and [qp1x] from on how to use execute_load.

4.50.1.2.2 execute_loaddc  With execute_loaddc statement any domain violation will be reported
and flagged as execution error. In contrast, the execute_load statement ignores all domain violations
and loads only data that meets the domain restrictions. In addition to loading data for sets, parameters
and variables, we can load a field of a variable into a parameter.

Warning: when loading a single field, all other fields are reset to their default value.

The statement with execute_loaddc can be used in one of the following forms:

execute_loaddc 'file_name', id1, id2, ..., idn ;
execute_loaddc 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;

where

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; read from the current working directory)</td>
</tr>
<tr>
<td>id1, id2, ..., idn</td>
<td>Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional</td>
</tr>
<tr>
<td>id1=gdxid1, id2=gdxid2, ..., id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional</td>
</tr>
<tr>
<td>setid=*</td>
<td>Allow to load the universe of labels from a GDX file into a set. Note, that only labels known to the GAMS program will be loaded.</td>
</tr>
</tbody>
</table>

See [load7] example on how to use execute_loaddc.

4.50.1.2.3 execute_loadpoint  The execute_loadpoint is similar to execute_load, however, the new
values are merged with the old values.

If no arguments besides the name of the GDX file are given, all variables and equations that match
variables and equations of the calling GAMS programs will be merged with the GDX level and marginal
values. Bounds, scales and priorities will remain unchanged.
If one or more symbols to be loaded are specified explicitly, values for these symbols will be merged. For
variables and equations only the levels and marginals will be merged. To merge other variable or equation
attributes like bounds, scales and priorities, they have to be listed explicitly (e.g. `execute_loadpoint
'file_name', x.lo;` to merge the lower bound of variable x).

The statement with `execute_loadpoint` can be used in one of the following forms:

```
execute_loadpoint 'file_name', id1, id2, ..., idn ;
execute_loadpoint 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;
```

where

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; read from the current working directory)</td>
</tr>
<tr>
<td>id1, id2, ..., idn</td>
<td>Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional</td>
</tr>
<tr>
<td>id1=gdxid1, id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file; note that commas are optional</td>
</tr>
</tbody>
</table>

### 4.50.1.3 Example 1 - Reading a GDX File

The file `trnsport.gms` (from `[TRNSPORT]`) has been modified to use the demand data from an
external source. Only the relevant declarations are shown.

The parameter B is read from the GDX file using the name 'demand', and only those elements that are in
the domain j will be used. Values for parameter B that are outside the domain j will be ignored without
generating any error messages.

```
* Example 1
Set
  j  markets / new-york, chicago, topeka / ;
Parameter
  B(j) demand at market j in cases ;
$gdxin demanddata.gdx
$load b=demand
$gdxin
```

### 4.50.1.4 Example 2 - Reading a GDX File

In this example, the set J is also read from the GDX file, and is used as the domain for parameter B. All
elements read for the set J will be used. Values for the parameter B that are outside the domain J will be
ignored. Note that the dimension of set J is set to one by specifying its domain.

```
* Example 2
$gdxin demanddata.gdx
Set
  J(*)  markets;
$load j=markets
Parameter
  B(j) demand at market j in cases ;
$load b=demand
$gdxin
```
4.50.1.5  Example 3 - Reading a GDX File

The $load command without parameters can read a listing of all symbols in a GDX file. The following:

* Example 3
$gdxin transport.gdx
$load

writes the following to the listing file:

Content of GDX C:\XLSFUN\TRANSPORT.GDX

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Dim</th>
<th>Count</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set</td>
<td>1</td>
<td>2</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>Set</td>
<td>1</td>
<td>3</td>
<td>j</td>
</tr>
<tr>
<td>3</td>
<td>Parameter</td>
<td>1</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>Parameter</td>
<td>1</td>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>Parameter</td>
<td>0</td>
<td>1</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>c</td>
</tr>
<tr>
<td>8</td>
<td>Variable</td>
<td>2</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>Variable</td>
<td>0</td>
<td>1</td>
<td>z</td>
</tr>
<tr>
<td>10</td>
<td>Equation</td>
<td>0</td>
<td>1</td>
<td>cost</td>
</tr>
<tr>
<td>11</td>
<td>Equation</td>
<td>1</td>
<td>2</td>
<td>supply</td>
</tr>
<tr>
<td>12</td>
<td>Equation</td>
<td>1</td>
<td>3</td>
<td>demand</td>
</tr>
</tbody>
</table>

which lists the items present by Type, Name, Number of sets the item is defined over(Dim), number of elements in the file for this item (Count).

4.50.1.6  Example 4 - Reading a GDX File

Sometimes, a set is implicitly given by the elements of a parameter symbol. For example,

parameter a(i) / seattle 350, san-diego 600 / ;

in transport.gms implicitly defines the set of plants i. GAMS does not allow us to provide domain checked data, if the data for domain sets is unknown. So this code produces a compilation error:

Set i plant;
Parameter a(i) capacity / seattle 350, san-diego 600 /;

When entering data directly in the GAMS source adding the domain sets before the actual parameter declarations is usually not a problem, but when data comes from external sources (e.g. spreadsheets, databases, etc), this often results in an additional query to the database, spreadsheet etc. Nowadays, such data exchange happens mostly via the GD facility. With the domain load capability of the compile time load instructions ($load, $loadDC, $loadR, $loadM, $loadDCM, and $loadDCR) one can project an index position from a parameter or set symbol in the GDX container and load this slice into a one dimensional set. Here is a simple example:
Set i plant;
Parameter a(i) capacity;
$gdxin data
$load i<adata a=adata

This will try to load set elements from the GDX parameter symbol adata into the set i and next load the GDX parameter adata into the GAMS parameter a. The latter one is no problem anymore, since the data for set i is known when loading symbol a. GAMS will use the domain information stored in GDX of parameter adata to identify the index position to project on. If no appropriate domain information can be found in GDX, the GAMS compiler will generate an error. In such case the user can explicitly select an index position (here first index position) from the GDX symbol:

$load i<adata.dim1 a=adata

The automatic index position matching (i.e. no .dimN) using the domain information stored in GDX matches on the name of the set to be loaded and the domain set names stored in GDX for the symbol. The domain in GDX are searched from right to left (start with n=symbol dimension, then n-1, n-2, ...) and stops at the first match. With the projection symbol <, the domain in GDX is searched from left to right. This follows the style of the GAMS run time projection operation:

option sym1<sym2, sym1<=sym2;

Here is an example how to load. The network is defined by the capacity parameter cap contained in a GDX container net.gdx:

parameter cap(n,n) / (1*3).4 5, 4.(5*9) 3 /;

The following code loads the entire node set n of the network as well as the nodes with outgoing (out) and incoming (in) arcs and the capacity c.

set n nodes, out(n), in(n);
parameter c(n,n) capacity;
$gdxin net
$loadM n<=cap n<cap
$loadDC out<cap.dim1 in<cap.dim2 c=cap
display n, out, in;

The listing file looks as follows:

```
---- 6 SET n nodes
  1,  2,  3,  4,  5,  6,  7,  8,  9
---- 6 SET out Domain loaded from cap position 1
  1,  2,  3,  4
---- 6 SET in  Domain loaded from cap position 2
  4,  5,  6,  7,  8,  9
```
There is a potential issue with loading domains from parameters that have a zero value for some record. Since GAMS works with sparse data, it is sometime difficult to distinguish between a record with value zero (0) and the non-existence of a record. This is usually not a problem since we know the domain of a parameter and hence know all potential records. In case of using a parameter to define the domain this represents a source of confusion. Moreover, GDX has the capability of storing true zeros (most GDX tools like gdxxrw have options (Squeeze=Y or N) to either write a true 0 or squeeze the 0s when writing GDX). So in case GDX has a zero record, a domain load from such a parameter will include this record. Here is an example. The spreadsheet Book1.xlsx contains the following data:

The GDX utility GDXXRW with the following command line:

```
gdxxrw Book1.xlsx Squeeze=N par=dat rng=Sheet1!a1 rdim=1
```

reads the Excel data and produces a GDX container Book1.gdx with a one dimensional parameter dat(*) which can be viewed in the GDX browser in the GAMSIDE:

Notice that label a4 is present while label a3 is not part of GDX symbol dat. Without the Squeeze=N (the default is Squeeze=Y) we also would not have seen a4. If we load dat to define the domain (remember we need to use $load i<dat.dim1 since gdxxrw does not write domain information to GDX), we will miss out on a3 but have a4 (assuming Squeeze=N). Please also note that the zero record disappears on regular loading and is turned into an EPS when loading under $OnEps:

```
set i;
parameter a(i);
$gdxin Book1
$load i<dat.dim1 a=dat
display i,a;
parameter a0(i);
$OnEps
$load a0=dat
display a0;
```

This results in a listing file

```
---- 5 SET i Domain loaded from dat position 1
  a1,   a2,   a4
---- 5 PARAMETER a
  a1 5.000,   a2 1.000
---- 9 PARAMETER a0
  a1 5.000,   a2 1.000,   a4  EPS
```

With gdxxrw parameter Squeeze=Y the listing file would look as follows:

```
---- 5 SET i Domain loaded from dat position 1
  a1,   a2
---- 5 PARAMETER a
  a1 5.000,   a2 1.000
---- 9 PARAMETER a0
  a1 5.000,   a2 1.000
```

4.50.1.7 Example 5 - Reading a GDX File

The following statement reads gams element k, d, f, a, b, and x from file trans2.gdx during execution phase:

```gams
execute_loadc 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

where k be renamed from j, a be renamed from sup, and b be renamed from dem in the GDX file tran2.gdx.

Suppose there is one element topeka missing from the set definition but the element remains in the GDX file tran2.gdx. As a consequences the listing file contains an error message like:

```
**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 1 Domain errors for symbol k
topeka

**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 2 Domain errors for symbol d
seattle.topeka
san-diego.topeka
```

and the job is aborted with an execution error.

Note

- domain errors occur whenever set element names are not spelled exactly the same as an element specified in the corresponding set in GAMS flagging alternative spellings or missing elements.
- domain errors do not arise when items are not specified with them set to zero (no entry for a set element leaves to a corresponding value of zero)

4.50.1.8 Inspecting contents of a GDX file

In addition to reading data from a GDX file during compile phase or execution phase there are a few ways to examine the contents of a GDX file.

- Inspecting contents with $load
- Inspecting contents with IDE
- Inspecting contents with GDXDUMP
- Inspecting contents with GDXDIFF

Inspecting contents with $load

The $load command without any parameters will show a listing of all symbols in the GDX file. See Example 3 - Reading a GDX File on how to use $load to get a listing of all symbols.

Inspecting contents with the IDE

The GAMSIDE can be used to view the contents of a GDX file by opening a GDX file with the Open file dialogue. The IDE only recognizes the .gdx file extension of a GDX file. The resultant display gives the names of the items in the GDX file on left hand part of the screen while the right part gives the exact data entries for the item highlighted in the left hand part.

For example, opening a GDX file using the Open file dialogue and and moving the cursor to the set i6 causes the screens, it becomes:

showing the elements in the set as contained within the GDX file. Similarly choosing the object modeistance yields the screen:

showing the data for this item. The data may be reordered via the mouse. For example placing the mouse on the column containing san francisco and dragging the column up yields:

which is the slice of the matrix containing san francisco data. In turn clicking on Brussels yields:
Note

• the columns are sortable in the left hand portion of the display. All one needs to do is to click on the gray boxes (Symbol, Type,...) with the mouse. There is also a search dialog at the bottom that permits one to look for select items.

• a right mouse click allows one to write the contents of any or all items to HTML format.

Inspecting contents with GDXDUMP

The GDXDUMP utility can list the symbols in the file and it also can write sets and parameters formatted as a GAMS data statement.

```
gdxdump gdxfile_name format=choice symb=optional choice
```

where

- the `gdxfile_name` is the name of the GDX file to write data from.
- the output is created to the screen not to a file. One can also write the GDX file contents into a GAMS file using the command:

```
gdxdump gdxfile_name > filetouse.gms
```

- Data for a selected set, parameter, variable or equation (under all three of the output options when a specific item is named using the SYMB option)
- Data for all sets, parameters, variables and equations (Under normal option when the SYMB is not used)
- Data on solution items (variables and equations) formatted in a fashion suitable for import as a basis in another GAMS program where the marginals and levels are output. All of the scalars, sets and parameters (tables) in a GDX file to standard output formatted as a GAMS program with data statements or in CSV format. It skips information for variables and equations.
- Under the format=CSV choice it only creates output when a symbol is selected using the SYMB syntax.
- Under the format=CSV choice when the requested symbol is a variable or an equation one only gets the level values not the marginal, under the other formats one gets all items.
- Under the format=gamsbas choice one gets all variables and equations when the SYMB syntax is not used.

Suppose we wish to write out the GDX file `tran.gdx`, then we would use the command:

```
gdxdump tran
```

See more Examples on inspecting contents with GDXDUMP.

Inspecting contents with GDXDIFF

The GDXDIFF utility can be used to compare two GDX files by creating a third GDX file containing a list of differences between all symbols. In particular for all items with the same name, type and dimension in the two GDX files the differences in numerical values are written to a third GDX file with A summary report written to standard output (ordinarily the LOG file).

This utility can be used either at command line, or by $Call, or execute command.
gdxdiff file1 file2 {diffile} {Eps = value} {RelEps = value} {Field = FieldName} {ID=Identifier}

GDXDIFIFF requires the first two file name parameters,

- File1 Name of the first GDX file
- File2 Name of the second GDX file

The remaining parameters are optional

- Diffile An optional third file name that is the name of the GDX file that contains the differences found in the parameters. If that parameter, is absent the file will be named 'diffile.gdx' and placed in the current directory.
- Eps = value A tolerance that is the maximum amount that two numbers may differ by i.e. given a1 and a2 then abs(a1-a2) is reported as different if it exceeds this tolerance
- RelEps = value A tolerance that is the maximum percentage amount that two numbers may differ by i.e. given a1 and a2 then abs(a1-a2)/max(abs(a1),abs(a2)) is reported as different if it exceeds this tolerance.
- Field = FieldName A field that if specified limits done between the information for variables and equations to specific attributes (Lo, L, Up, M, Scale and Prior)
- ID=Identifier Limits the comparisons to selected items; items not specified will be ignored. Multiple items can be specified as: ID=id1 ID=id2 or ID="id1 id2"

Suppose we wish to compare the GDX files tran.gdx and tran2.gdx, then we would use the command:

```
gdxdiff tran tran2
```

In turn the output to standard output (nominally the terminal screen) appears as follows:

```
Summary of differences:
  d  Data is different
  dem  Keys are different
  sup  Keys are different
supply  Symbol not found in file 1
  x  Symbol not found in file 1
```

and summarizes the differences found. Simultaneously the file diffile.gdx when examined in the IDE contains the following

which reports on the differences found in the two files.

Note

- Some new coding is introduced in the difference GDX file. Namely a new dimension is added to the parameters being compared which can contain 4 entries
  - dif1 indicates that the entry occurs in both files and shows the value found in the first file.
  - dif2 indicates that the entry occurs in both files and shows the value found in the second file.
  - ins1 indicates that the entry only occurs in the first files and shows the value found.
  - ins2 indicates that the entry only occurs in the second file and shows the value found.
- Only named items with the same name, type and dimension will be compared in the diffile.gdx output. Named items that are new or are deleted will only appear in the standard output summary report.

See more Examples on inspecting contents with GDXDIFFF.
4.50.2 Writing a GDX file

Writing of GDX files in a GAMS model can be done during the compile phase or the execution phase. A GDX file can also be written after compilation and execution.

4.50.2.1 Compile Phase

During compilation, we can use a group of Dollar Control Options to write data to a GDX file. Writing during the compilation phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. The following Dollar Control Options are available for writing data to a GDX file into GAMS during compilation of a GAMS model:

- $gdxout
- $unload

4.50.2.1.1 $gdxout

The $gdxout command is used in a sequence either to unload specified items to a GDX file or to close the specified GDX file. The statement with $gdxout can be used in one of the following forms:

```
$gdxout file_name
$gdxout
```

where:

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; written to from the current working directory)</td>
</tr>
<tr>
<td></td>
<td>(no file_name) closes the GDX file</td>
</tr>
</tbody>
</table>

See Example 1 - Writing a GDX file and [CompTimeWriteTranspGDX] for examples on how to use $gdxout.

4.50.2.1.2 $unload

The $unload command unloads specified items to the GDX file. The statement containing $unload is preceded and succeeded by the $gdxout statements where

- the preceding $gdxout specifies the GDX file name and opens the file.
- the succeeding $gdxout closes the GDX file.

The statement with $unload command can be used in one of the following forms:

```
$unload
$unload id1 id2 ... idn
$unload id1=gdxid1 id2=gdxid2 ... idn=gdxidn
```

where:
<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(no identifiers)</td>
<td>Write all symbols to the gdx file</td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Write GAMS symbols id1, id2, ..., idn to the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2 ... idn=gdxidn</td>
<td>Write GAMS symbols id1, id2, ..., idn with corresponding names gdxid1, gdxid2, ..., gdxidn into the GDX file</td>
</tr>
</tbody>
</table>

See Example 1 - Writing a GDX file and [unload1] - [unload9] for examples on how to use $unload. See also [CompTimeWriteTrnsportGDX] example.

### 4.50.2.2 Execution Phase

When writing data to a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to write data to a GDX file during execution phase are

- `execute_unload`
- `execute_unloaddi`
- `execute_unloadidx`
- `savepoint`

#### 4.50.2.2.1 `execute_unload`, `execute_unloaddi`, and `execute_unloadidx`  
The `execute_unload` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file. Without specifying any identifier, all sets, parameters, variables and equations will be written to the GDX file.

The `execute_unloaddi` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file similar to `execute_unload`, but also writes the domains of all unloaded symbols to the same file.

The `execute_unloadidx` statement requires that each symbol written is a parameter; each parameter must have a domain specified for each index position. These domains have the requirement that they are formed using an integer sequence for the UELs that starts at 1 (one). The domain names are changed to indicate the size of each domain. This information is used when reading the data back from the GDX file using `$LoadIDX` during compilation. Using the special domain names, the UELs for the domains can be recovered without writing the domains to the GDX file; see example below.

The statement with `execute_unload` can be used in one of the following forms:

```gams
execute_unload 'file_name' , id1, id2, ..., idn ;
execlineexecute_unload 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execlineexecute_unload 'file_name' , setid=* ;
execline```

The statement with `execute_unloaddi` can be used in one of the following forms:
The statement with `execute_unloaddidx` can be used in one of the following forms:

```sql
execute_unloaddidx 'file_name', id1, id2, ..., idn;
execute_unloaddidx 'file_name', id1=gdxid1, id2=gdxid2, ... ;
execute_unloaddidx 'file_name', setid=* ;
```

where
### 4.50 GAMS Data eXchange (GDX)

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Specify the name of the GDX file (with or without the extension .gdx; written to from the current working directory)</td>
</tr>
<tr>
<td>id1, id2, ..., idn</td>
<td>Write GAMS symbols id1, id2, ..., idn into the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1, id2=gdxid2</td>
<td>Write GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 into the GDX file</td>
</tr>
</tbody>
</table>

**Note**
- when only `file_name` is specified without other parameters all GAMS Symbols will be written into the GDX file `file_name`.
- The GAMS option `gdxUELs` controls which UELs are registered in `file_name`. With option `gdxUELs = squeezed;` (default) only the UELs that are required by the exported symbols are registered while all known UELs are registered if we set option `gdxUELs = full;`. See also [unload10].

See Example 2 - Writing a GDX file and [qp1x] on how to use `execute_unload`. See [unload10] on how to use `execute_unloadidx`. See Example 3 - Writing a GDX file on how to use `execute_unloadidx`.

#### 4.50.2.2.2 Savepoint
A GDX file containing the marginals and levels for all variables and equations at the end of a solve will be created with the command line parameter, model attribute or option statement `Savepoint`. One can save the solution information from the last solve or from every solve. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point.

The basic command line form is:

```plaintext
gams mymodelname Savepoint=number
```

the model attribute form is

```plaintext
modelname.savepoint=number;
```

and the option statement form is

```plaintext
option savepoint=number
```

where

- when number equals 1/3 a point gdx file is saved from the last solution collected in the GAMS model and the file name will be `modelname_p.gdx` where model name is the name of the model identified in the solve statement.
- when number equals 2/4 a point gdx file is saved from the every solve in the GAMS model and the file name will be `modelname_pnn.gdx` where model name is the name of the model identified in the solve statement and nn is the internal number of the solve. Thus if 10 solves occur one will get 10 files named `modelname_p1.gdx` through `modelname_p10.gdx`. 
Note

When solving \textit{asynchronously}, one should keep in mind, that the internal numbers mentioned for \texttt{savepoint=2/4} are generated when submitting the model, not when collecting it. And with \texttt{savepoint=1/3} it is the last solution collected, not the last model submitted that is written to the savepoint.

The following example:

\begin{verbatim}
model firm / all /;
    firm.savepoint=1;;
solve firm using LP maximizing objfun;

saved a point gdx file \texttt{firm.p.gdx}.

and:

model transport /all/ ;
    option savepoint=2;
    set newseattle /s1,s2/;
    parameter datador(newseattle) /s1 350, s2 450/;

    loop(newseattle,
        a("seattle")=datador(newseattle);
        Solve transport using lp minimizing z ;
    );
    Display x.l, x.m ;

saved two point gdx files \texttt{transport.p1.gdx} and \texttt{transport.p2.gdx}.
\end{verbatim}

\subsection*{4.50.2.3 Example 1 - Writing a GDX file}

This example has modified the file \texttt{trnsport.gms} from [TRNSPORT] by adding the following statements after the last line.

\begin{verbatim}
...[TRNSPORT]...

d(i,j)=d(i,j)*10;

$gdxout tran
$unload i j
$unload d
$unload f
$unload b=dem a=sup
$gdxout
\end{verbatim}

This example creates the GDX file \texttt{tran.gdx} containing the data for the sets \texttt{i} and \texttt{j} as well as the parameters \texttt{d}, \texttt{f}, \texttt{a} and \texttt{b} during the compile time, when \texttt{a} and \texttt{b} have been renamed in the GDX file to \texttt{dem} and \texttt{sup}. Also note the parameter \texttt{d} will not have been multiplied by 10 but rather take on their compile time value.

An \texttt{$unload} statement above can specify more than one symbol to be written to a GDX file and the similar result could also be accomplished using:

\begin{verbatim}
...[TRNSPORT]...

d(i,j)=d(i,j)*10;

$gdxout tran
$unload i j d f b=dem a=sup
$gdxout
\end{verbatim}
4.50.2.4 Example 2 - Writing a GDX file

This example has modified from the file trnsport.gms (from [TRNSPORT]) by adding the following statement right after the solve statement.

...[TRNSPORT]...

Solve trnsport using LP minimizing Z;
execute_unload 'results.gdx', i, j, z, x;

After solving the model, the sets i and j and the variables z and x with all the data available after the solve.

4.50.2.5 Example 3 - Writing a GDX file

This example shows the use of the indexed write and read data during execution phase:

Set I /1*100/,
  J /1*50 /
parameter A(I,J) /1.1=11, 1.9=19, 10.1=101/
execute_unloadidx 'data.gdx', A;

Viewing the file data.gdx in the gamside shows the modified domain information:

To read from data.gdx, we use the indexed read:

Set I,J;
parameter A(I,J);
* load the data
$gdxin data.gdx
$loadidx A
$gdxin
* write all symbols so we can inspect in the gamside
$gdxout test.gdx
$unload
$gdxout
execute_unloadidx 'data.gdx', A;

Viewing the file test.gdx in the gamside shows that the domains have been populated:
4.50.2.6 Writing a GDX file after compilation or execution

A GDX file containing all data items resident at the end of the run of a GAMS code can be created using the gdx command line option either via GAMS call at the command line or via the command line parameter box in the IDE. This will cause all sets, parameters, variables and equations to be written to the GDX file.

For example:

```
gams mymodelname gdx=gdxfile_name
```

Or

```
gams mymodelname action=c gdx=gdxfile_name
```

where

- `mymodelname` specifies the name of the GAMS file
- `gdxfile_name` gives the file name and possible path where the GDX file is to be retained. When no path is specified the default directory is the current working directory or project directory in the IDE as seen below.
- `action=c` indicates request to write a GDX file after compilation only
- setting GDX to the string "default" (i.e. `gdx=default`) causes GAMS to create a GDX file with the GAMS file root name and a GDX extension. Thus

```
gams transport gdx=default
```

will cause GAMS to write the gdx file `transport.gdx`.

When the IDE is used, the GDX file creation is invoked by an entry in the upper right hand corner of the IDE screen as illustrated below:

Note

- When this option is used the GDX file is created just at the end of the GAMS execution so the data written will contain the current values for all sets, parameters, variables and equations that are on hand at the end of the GAMS job.
- The GDX data for the variables and equations contains the levels, marginals, lower bounds, upper bounds and scales for each item.
- This yields a file that may be automatically opened in the IDE by doing a mouse click on the highlighted line in the IDE process window.

Using the gdx command line parameter when running the model using the IDE, the process window will show the GDX file name in blue indicating that the file can be opened using a double-click with the mouse. See also Inspecting contents with IDE.
4.50.3 General notes on GDX files

There are several things worth noting about GDX files:

- When working with GDX only one GDX file can be open at a time.
- When reading data from a GDX file, the symbol to be read must be declared before the reading statement appears.
- When the GDX file to be written has the same name as an existing GDX file the existing file will be overwritten. The resultant file will only contain the new data; there is no merge or append option.
- The \$unload command to write a GDX during compile time will only write out data defined in the compilation at the point where the command appears. No results of any solves or calculations done within the current GAMS program will be reported with \$unload. This is not true when using the execute_unload or execute_unloaddi commands.
- Both execute_unload and execute_unloaddi will write out data defined in the execution sequence at the point where the command appears. The results of the most recent solve command and any parameter calculations occurring before the GDX write will be reported.
- Any subsequent execute_unload or execute_unloaddi to a GDX file written earlier will totally overwrite that file so care must be taken to write all wanted information in the last appearing execute_unload or execute_unloaddi.
- A command line GDX write using the gdx=file\_name command line parameter will write out data defined at the end of the execution sequence. The results of the most recent solve and any parameter calculations will be reported.
- When loading data domain checking will not be enforced. When items are resident in the GDX file for set elements that are not present in the current file these items will be ignored. GAMS will not generate any message telling you which items are ignored.
- Options Savepoint and execute_Loadpoint provide a GDX way of saving and loading a basis.
- The contents as they differ between GDX files can be examined with GDXMERGE or GDXDIFF.
- Starting with GAMS version 22.3, gdx files can be written in a compressed format unless the environment variable GDXCOMPRESS is set to zero. A value of 1 indicates compression.
- GDX files from a different GAMS version can possibly be incompatible due to compression among other changes. A current GAMS system can read all older GDX file formats. GDX files can be converted to a compressed format or an older format using GDXCOPY.
- Users can also write their own programs accessing data in a GDX file via the expert-level GDX API using gdxdclib library. The interface and usage for this library is described in a separate document; see gdxioapi.chm or gdxioapi.pdf.

4.50.4 GAMS Data eXchange Tools

A number of GDX based tools and GDX related tools are included in the GAMS distribution and maintained by GAMS. See also Tools Manuals.

- The data exchange tools provide functionality to exchange data between GAMS and other data sources.
- The GDX service tools operate directly on GDX containers.
- Some of Data transformation tools perform very specific data transformation tasks that are either awkward or inefficient to implement in GAMS directly.

Some utilities are available only on specific platform. See Supported Platforms for more details.
4.51 Extended Mathematical Programming (EMP)

Extended Mathematical Programming (EMP) is an extension to algebraic modeling languages that facilitates the automatic reformulation of new model types as models in more established mathematical programming classes, allowing them to be solved by mature solver algorithms. A number of important problem classes can be solved, e.g. Nash games and equilibria, bilevel programs, Disjunctive Programs and Stochastic Programs. EMP is independent of the modeling language used but is currently implemented only in GAMS. The new types of problems modeled with EMP are reformulated, using the GAMS solver JAMS, as well established types of problems, and the reformulated models are passed to a suitable GAMS solver to be solved.

EMP models are defined by information taken from two places: the traditional model definition and a text file containing annotations or additional information. The usual model definition contains variables, constraints and/or functions, and perhaps also an objective or matching information. Additional annotations to specify relationships that don't fit within this traditional definition are taken from the EMP info file. For example, the constraints for two optimizing agents in a competitive game can be specified with traditional algebra, while the structure (who owns what variables and constraints) can be specified in the info file. Together, this allows large, complex models to be specified in a convenient, precise, and flexible way.

This chapter is organized as follows.

- EMP Annotations: the EMP Info File
- Soft Constraints
- Variational Inequalities (VI)
- Quasi-Variational Inequalities (QVI)
- Equilibrium Problems
- Embedded Complementarity Systems
- Equilibrium Problems with Shared Constraints
- Equilibrium Problems with Shared Variables
- Bilevel Programs
- Disjunctive Programming
- Stochastic Programming
- EMP Keywords

Note

- At the end of each section we present and discuss the general syntax that EMP provides to write the annotations for that particular problem type. We use the usual GAMS syntax symbols: [ ] (the enclosed construct is optional), { } (the enclosed construct may be repeated zero or more times) and | (exclusive OR).
- Many EMP model examples are available in the GAMS EMP Library.
4.51 Extended Mathematical Programming (EMP)

4.51.1 EMP Annotations: the EMP Info File

EMP models are defined by both the usual content of a GAMS model and annotations found in a simple text file named emp.info (aka the EMP info file). It is often most convenient to create this file via the GAMS put writing facility. The annotations primarily serve to define the model (e.g. to specify that a variable $u$ is really the dual multiplier for a constraint $g$) but can also specify how a solver should process the model. The annotations make use of EMP keywords to do this.

A simple example will serve as illustration. Consider the following NLP:

$$\begin{align*}
\min_{x,y,z} & -3x + xy \\
\text{s.t.} & x + y \leq 1 \\
& x + y - z = 2 \\
& x, y \geq 0
\end{align*}$$

We will use EMP annotations to automatically generate the first order conditions (KKT conditions) of this NLP and thus reformulate the NLP as an MCP:

Variables $f$, $z$;
Positive Variables $x$, $y$;

Equations $g$, $h$, defobj;

$g.. \quad x + y \leq 1$
$h.. \quad x + y - z = 2$
$defobj.. \quad f =e= -3x + x*y$

Model comp / defobj, g, h /

File info / '%emp.info%' /
putclose info / 'modeltype mcp';
solve comp using EMP minimizing f;

Observe that the model is defined in the usual way and the file emp.info contains just one line: modeltype mcp. The EMP keyword modeltype indicates that the value following the keyword is the model type to be used for the reformulation. In this example the model type is mcp. Here this specification is required: the sole point of our EMP annotations is to generate an MCP and not (as is usually the case) to define the model. Usually, the model algebra and annotations together imply the type of the reformulated model and so no modeltype specification is required or wanted. Finally, note that the model type in the solve statement is EMP: this is typical.

The solver JAMS implements the EMP framework. It processes the model and the annotations, automatically reformulates the original EMP model as a model of a different (more easily solved) type, passes the reformulated model on to an appropriate subsolver, and maps the resulting solution back into the original problem space.

In case users wish to inspect the (scalar) reformulated model, the JAMS option FileName may be used to specify the name of the file containing this model. Adding the following lines before the solve statement in the GAMS code above will cause the MCP reformulation to be saved in the file myReform.gms.

File empopt / 'jams.opt' /
comp.optfile = 1;
putclose empopt / 'FileName myReform.gms';

The listing file will contain some additional information - the EMP Summary - as part of the output for each EMP model solved. We provide details on the EMP summary for each reformulation that we discuss below.
4.51.2 Soft Constraints

In many cases modelers wish to relax certain constraints: violation of these constraints is allowed but is associated with a well-defined penalty. The constraints that are allowed to be violated are called *soft constraints* and the constraints that continue to hold are called *hard constraints*.

In this section we present a mathematical formulation of soft constraints, give an example of how soft constraints can be modeled with GAMS EMP and introduce the EMP annotations specific to soft constraints.

4.51.2.1 Soft Constraints: Mathematical Formulation

A general formulation of a constrained minimization problem is:

\[
\begin{align*}
\text{Min} & \quad f(x) \\
\text{s.t.} & \quad c_i(x) \leq 0, \quad \forall i \in \mathcal{I},
\end{align*}
\]  

(2)

where \( f \) and the functions \( c_i \) are smooth, real-valued functions on a subset of \( \mathbb{R}^n \) and \( \mathcal{I} \) is a finite set of indices. Note that in this problem all feasible solutions must satisfy all constraints \( c_i \).

Now assume that some constraints are allowed to be violated (i.e. made soft), while the remaining constraints continue to hold for all feasible solutions. The soft constraints are associated with a penalty function that is added to the objective function. Since we have a minimization problem, the effect will be a balance or compromise between competing goals: minimizing the objective and minimizing the penalty functions.

\[\text{Note}\]

The penalty terms for soft constraints are *added* to the objective function in a minimization problem and *subtracted* in a maximization problem.

Let \( \mathcal{L} \in \mathcal{I} \) be the set of indices for the soft constraints: this implies \( \mathcal{M} := \mathcal{I} \setminus \mathcal{L} \) is the index set for the hard constraints. Problem (2) becomes:

\[
\begin{align*}
\text{Min} & \quad f(x) + \sum_{i \in \mathcal{L}} w_i g_i(c_i(x)) \\
\text{s.t.} & \quad c_i(x) \leq 0, \quad \forall i \in \mathcal{M},
\end{align*}
\]  

(3)

where the penalty functions \( g_i \) are real-valued functions of \( c_i(x), i \in \mathcal{L} \). As we will see later, EMP has implementations of the following penalty functions: absolute value, least squares, and the maximum of a term and zero. Further, \( w_i \) is a multiplier associated with each penalty term, also called the *weight*. The weights facilitate prioritizing soft constraints: the weight of more important constraints will be greater than the weight of lesser constraints.
4.51.2.2 Soft Constraints with EMP: A Simple Example

The following simple example is adapted from the JAMS solver manual:

\[
\begin{align*}
\text{Min} & & -x_1^2 \\
\text{s.t.} & & \log(x_1) = 1 \\
& & x_2^2 \geq 2 \\
& & 3x_1 + x_2 \leq 5 \\
& & x_1, x_2 \geq 0 
\end{align*}
\]  \hspace{1cm} (4)

This problem can be formulated in GAMS as follows:

Positive Variables x1, x2;
Variables obj;

Equations f0 "objective function", f1, f2, f3;

f0.. obj =e= -sqr(x1);
f1.. log(x1) =e= 1;
f2.. sqr(x2) =g= 2;
f3.. 3*x1 + x2 =l= 5;

Model m /all/;
x1.l = 1; x2.l = 1;
solve m using NLP min obj;

Note that this problem has no feasible solution. Thus we choose to relax the first two constraints by adding a penalty for their violation to the objective function. We also weight the relative importance or priority of the objective and the violations of these two constraints by introducing weights to go with these penalty functions. The resulting problem reads as follows:

\[
\begin{align*}
\text{Min} & & -x_1^2 + 5 \|\log(x_1) - 1\|^2 + 2 \max(x_2^2 - 2, 0) \\
\text{s.t.} & & 3x_1 + x_2 \leq 5 \\
& & x_1, x_2 \geq 0
\end{align*}
\]  \hspace{1cm} (5)

Note that the first constraint is replaced by a least squares penalty of \((\log(x_1) - 1)\) with a weight of 5 and the second constraint by the penalty term \(\max(x_2^2 - 2, 0)\) with a weight of 2. However, the "max" penalty makes the objective function non-smooth. To rectify this, we implement the "max" penalty by introducing a new variable \(v\) that, due to its bounds and the direction of optimization, will take the value \(\max(x_2^2 - 2, 0)\) at the solution. The result is:

\[
\begin{align*}
\text{Min} & & -x_1^2 + 5 \|\log(x_1) - 1\|^2 + 2v \\
\text{s.t.} & & 3x_1 + x_2 \leq 5 \\
& & x_1, x_2 \geq 0 \\
& & v \geq x_2^2 - 2 \\
& & v \geq 0
\end{align*}
\]  \hspace{1cm} (6)

This reformulation could of course be implemented directly in GAMS using standard GAMS syntax, but the EMP solution is easier to create and maintain, to read and understand, and to scale upwards as problem size and complexity increase. For the latter, we can simply add the following EMP annotations and updated solve statement (to specify solution as an EMP) to the GAMS code above:
File soft / ’%emp.info%’ /;
put soft;
$onput
adjustequ
f1 sqr 5
f2 maxz 2
$offput
putclose;
solve m using EMP min obj;

Note that the EMP annotations contain three lines: the first line, containing the EMP keyword adjustequ , indicates that the lines that follow specify soft constraints, i.e. equations to be converted to penalty terms; the second line specifies the name of the first soft constraint f1, the penalty function to use, and - optionally - the weight; and similarly the third line specifies the name of the second soft constraint f2, the penalty function to use, and the weight.

The solver JAMS will use the information in the EMP annotations to automatically reformulate problem (4) as problem (6) and pass it along to an NLP subsolver. The EMP Summary produced by JAMS will contain the following line:

--- EMP Summary
   ...
   Adjusted Constraint = 2
   ...

This output reflects that indeed two constraints were ”adjusted”, i.e. converted to penalty terms in the objective function.

4.51.2.3 EMP Syntax for Soft Constraints

The EMP framework provides the following general syntax to specify constraints that are converted to soft constraints:

\[
\text{AdjustEqu } \text{equ } \text{abs|sqr|maxz } \{\text{weight}\}
\{\text{equ abs|sqr|maxz } \{\text{weight}\}\}
\]

The EMP keyword AdjustEqu indicates that the lines that follow specify equations that are ”adjusted”: they are moved from constraints to penalty terms in the objective function. Equ is the name of the equation to penalize, while the three EMP keywords that follow indicate the penalty function to use. If not specified, the weight used defaults to 1. For an example, see above or the model [SIMPENLP] in the GAMS EMP Model Library.

4.51.3 Variational Inequalities (VI)

Variational inequalities provide a general mathematical framework for many problems arising in optimization. For example, constrained optimization problems like LP and NLP are special cases of VI, and systems of equations and complementarity problems can be cast as VI. Thus VI problems have many applications, including those in transportation networks, signal processing, regression analysis, and game theory.

In this section we present a mathematical formulation of VI, give an example of how VI can be modeled with GAMS EMP, and introduce the EMP annotations specific to VI. Note that to fully and properly understand some of this section, the introduction to MCP provides a useful background.
4.51.3.1 Variational Inequalities: Mathematical Formulation

For a given continuous function \( F: \mathbb{R}^n \to \mathbb{R}^n \) and a fixed closed convex set \( K \subset \mathbb{R}^n \) the variational inequality problem \( VI(F, K) \) is to find a point \( x^* \in K \) such that:

\[
\langle F(x^*), (x - x^*) \rangle \geq 0, \quad \forall x \in K, \tag{7}
\]

where \( \langle \cdot, \cdot \rangle \) denotes the usual inner product.

Observe that the VI generalizes many problem classes:

- If \( F(x) = 0 \) and \( K \equiv \mathbb{R}^n \), the VI is a system of nonlinear equations.
- If \( F(x) = \nabla f(x) \), the VI is a convex optimization problem.
- If \( F(x) = \nabla f(x) \) and \( K = \{ x \mid Ax = b, Hx \leq h \} \), the VI is an NLP.
- If \( F(x) = \nabla f(x) = p \) and \( K = \{ x \mid Ax = b, Hx \leq h \} \), the VI is an LP.
- If the feasible region is a box \( B = \{ x \in \mathbb{R}^n \mid l_i \leq x_i \leq u_i, \text{ for } i = 1, \ldots, n \} \), with \( l_i \leq u_i \) \( l_i \in \mathbb{R} \cup \{-\infty\} \) and \( u_i \in \mathbb{R} \cup \{\infty\} \), the VI is an MCP, where \( x^* \in B \) is a solution of the respective MCP if for each \( i = 1, \ldots, n \) one of the following conditions holds:

\[
\begin{aligned}
F_i(x^*) &= 0 \quad \text{and} \quad l_i \leq x^*_i \leq u_i, \\
F_i(x^*) &> 0 \quad \text{and} \quad x^*_i = l_i, \\
F_i(x^*) &< 0 \quad \text{and} \quad x^*_i = u_i.
\end{aligned} \tag{4.18}
\]

Note that the set \( K \) is frequently defined in the following way:

\[
K = \{ x \mid x \geq 0, h(x) \geq 0 \}. \tag{8}
\]

Further, note that the \( VI(F, K) \) represents a wider range of problems than classical optimization whenever \( F(x) \neq \nabla f(x) \) for some objective function \( f \) (or equivalently, the Jacobian of \( F \) is not symmetric). For example, problems that can be cast as VI include (generalized) Nash games and Nash equilibrium problems, systems of equations, complementarity problems, and fixed-point problems.

4.51.3.2 Variational Inequalities with EMP: A Simple Example

Consider the following simple three dimensional linear example (adapted from Yashtini & Malek (2007) [207]). Let

\[
F(x) = \begin{bmatrix}
22x_1 - 2x_2 + 6x_3 - 4 \\
2x_1 + 2x_2 \\
6x_1 + 2x_3
\end{bmatrix}, \quad K = \{ x \in \mathbb{R}^3 \mid x_1 - x_2 \geq 1, -3x_1 - x_3 \geq -4, 2x_1 + 2x_2 + x_3 = 0, l \leq x \leq u \}, \tag{9}
\]

where \( l = (-6, -6, -6)^T \), \( u = (6, 6, 6)^T \). N.B.: \( F \) is not the gradient of any function \( \mathbb{R}^3 \to \mathbb{R} \). This \( VI(F, K) \) has a unique solution: \( x = (2/3, -1/3, -2/3) \). The problem can be implemented in GAMS with EMP as follows:
Set  i /1*3/;
Variable x(i);

x.lo(i) = -6;

x.up(i) = 6;

Equations F(i), h1, h2, h3;

F(i).. (22*x('1') - 2*x('2') + 6*x('3') - 4)$sameas(i,'1')
+ (2*x('1') + 2*x('2'))$sameas(i,'2')
+ (6*x('1') + 3*x('3'))$sameas(i,'3')

= = 0;

h1.. x('1') - x('2') = = 1;

h2.. -3*x('1') - x('3') = = -4;

h3.. 2*x('1') + 2*x('2') + x('3') = = 0;

Model linVI / F, h1, h2, h3 /;

File annotations /'%emp.info%'/;

put annotations;
putclose 'vi F x h1 h2 h3';

solve linVI using EMP;

Observe that the function $F$ and the constraints $h$ are formulated using standard GAMS syntax. $F$ is implemented as an equation of type $=n=$, which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The annotations in the EMP info file define the structure of the VI: what functions are matched to what variables, and what constraints serve to define the set $K$. The EMP keyword $vi$ indicates that the model is a VI, that the VI function $F$ is matched to the variable $x$, and that the constraints $h1$ $h2$ $h3$ define the set $K$.

Alternatively, the EMP annotations could be written as follows:

putclose 'vi F x';

Here the equations after the equation-variable pair are omitted. This is acceptable, since by default any equations that are part of the model but are not matched with a variable are automatically used to define the set $K$.

Since VI problems have no objective, the short form of the solve statement is used.

The solver JAMS will reformulate the VI as an MCP and pass this on to an MCP subsolver. The EMP Summary produced by JAMS will contain the following line:

--- EMP Summary
...  
  VI Functions = 3
... 

This output reflects the fact that there were three VI functions in the model above, one for each member of the set $i$.

Note that there are several VI models in the GAMS EMP Library. For example, the models [SIMPLEVI] and [VI_MCP] demonstrate how some models can be specified using either MCP or VI syntax. A simple nonlinear VI is given in model [SIMPLEV12]. As the transportation model is so well known, model [TRANSVI] demonstrates how it can be cast as a VI.
4.51.3.3 EMP Syntax for Variational Inequalities

The general syntax of the EMP annotations used to specify variational inequalities is as follows:

\[
\text{VI } \{\text{var}\} \{ [\text{-}] \text{ equ var} \} \{ [\text{-}] \text{ equ} \}
\]

The EMP keyword \text{VI} indicates that this is a variational inequality specification. The core of the VI specification is the (list of) equation-variable pair(s): the other parts are optional. A pair matches the equation \text{equ} with the variable \text{var}. This indicates that \text{equ} defines part of the VI function \(F\), and that these rows of \(F\) are perpendicular to columns taken from \text{var}. Multiple equation-variable pairs are allowed. The optional variables before the pairs are called \text{preceding variables}. These are variables that appear (and are often defined by) the constraints of the model, but they are not matched explicitly via the VI function \(F\). Instead, they are automatically matched with the zero function. See model [ZEROFUNC] for an example and a more detailed discussion. The optional equations after the equation-variable pairs are called \text{trailing equations}. They define the set \(K\) and may be omitted from the VI specification. By default, any equations that are included in the model but are not matched with a variable are automatically used to define the set \(K\). Even though both preceding variables and trailing equations may be omitted from the VI specification, we recommend to explicitly list them, since this clarifies intentions and eliminates ambiguity.

The \(^{-}\) sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that \(x^{\ast\ast}1.5 \leq y\) becomes \(y \geq x^{\ast\ast}1.5\). Flipped equations in EMP behave in the same way as flipped equations in MCP.

Note

More than one VI specification may appear in a model. Often, it makes no difference whether multiple \text{equ-var} pairs are part of the same or separate VI specifications, but this is not the case in general. For an example, see model [SIMPLEVI4].

4.51.4 Quasi-Variational Inequalities (QVI)

Quasi-variational inequalities are a generalization of the variational inequality model: in a VI, the feasible set is fixed, while the QVI allows the feasible set to vary with or be a function of the variables in the model. To avoid repetition, we assume you are already familiar with the theory and notation for VI models. In this section, we present a mathematical formulation of QVI, give an example of how QVI can be modeled with GAMS EMP, and introduce the EMP annotations specific to QVI.

4.51.4.1 Quasi-Variational Inequalities: Mathematical Formulation

For a given continuous function \(F : \mathbb{R}^n \to \mathbb{R}^n\) and a point-to-set mapping \(K(y) : \mathbb{R}^n \to \mathbb{R}^n\), the quasi-variational inequality problem \(QVI(F, K)\) is to find a point \(y^{\ast} \in K\) such that:

\[
\langle F(y^{\ast}), (y - y^{\ast}) \rangle \geq 0, \quad \forall y \in K(y^{\ast}),
\]

(4.19)

where \(\langle \cdot, \cdot \rangle\) denotes the usual inner product.

If the point-to-set mapping \(K(y) : \mathbb{R}^n \to \mathbb{R}^n\) is constant, then the QVI above reduces to a VI. In order to define \(K(x)\) as a function of \(x\) it is convenient to introduce a shadow copy \(x\) of the variable \(y\). It should be understood that we have only one variable that appears in two forms: the variable of interest \(y\) and the parameter variable \(x\). Where \(y\) appears, we have a variable in the usual sense, and we take derivatives wrt this variable when deriving optimality conditions. Where \(x\) appears, we have a constant variable, i.e. we assume it is fixed when deriving optimality conditions. We can now use a function \(g(y, x)\) to define \(K(\cdot)\) and express QVI as: find a point \(y^{\ast} \in K\) (with its associated parameter variable \(x^{\ast}\)) such that:

\[
\langle F(y^{\ast}), (y - y^{\ast}) \rangle \geq 0, \quad \forall y \text{ s.t. } g(y, x) \leq 0.
\]

(4.20)
4.51.4.2 Quasi-Variational Inequalities with EMP: A Simple Example

Consider the following simple two dimensional linear example. Let $y$ be the variable of interest with its parameter variable $x$ and let

$$F(y) = \begin{bmatrix} 2y_1 + \frac{2}{3}y_2 - \frac{100}{22.5} \\ 1.25y_1 + 2y_2 - \frac{100}{22.5} \end{bmatrix}, \quad K(x) = \{y \in \mathbb{R}^2 | y_1 + x_2 \leq 15, x_1 + y_2 \leq 20, 0 \leq y, x \leq 11\}. \quad (4.21)$$

This $QVI(F, K)$ has a solution $y = (10, 5)$. The problem can be implemented in GAMS with EMP as follows:

```gams
set i / 1*2 /;
alias(i,j);
table A(i,j)
  1 2
  1 2 [8/3]
  2 [5/4] 2 ;
parameters
  b(i) / 1 [100/3], 2 22.5 /
  Cy(i,j) / 1.1 1, 2.2 1 /
  Cx(i,j) / 1.2 1, 2.1 1 /
  rhs(i) / 1 15, 2 20 /
; positive variables
  y(j) 'variable of interest, aka decision variable'
  x(j) 'parameter variable shadowing y'
  ; y.up(j) = 11; x.up(j) = 11;
equations
  F(i) 'FOC for agent optimization models'
  g(i) 'define feasible set K(x) for QVI'
  ;
  F(i).. sum{j, A(i,j)*y(j)} - b(i) =N= 0;
  g(i).. sum{j, Cy(i,j)*y(j)} + sum{j, Cx(i,j)*x(j)} =L= rhs(i);
model m / F, g /;
file annotations / '%emp.info%' /;
putclose annotations 'qvi F y x g' ;
solve m using emp;
```

Observe that the function $F$ and the constraints $g$ are formulated using standard GAMS syntax. $F$ is implemented as an equation of type $=n=$, which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The annotations in the EMP info file define the structure of the QVI: what functions are matched to what variables, and what constraints serve to define the mapping $K(x)$. The EMP keyword $qvi$ indicates that the model is a QVI, that the QVI function $F$ is matched to the variable-of-interest $y$ with its parameter variable $x$, and that the constraints $g$ define the mapping $K(x)$.

Since QVI problems have no objective, the short form of the solve statement is used.

The solver JAMS will reformulate the QVI as an MCP and pass this on to an MCP subsolver. The EMP Summary produced by JAMS will contain the following lines:
--- EMP Summary

...  
VI Functions = 2
QVI Parameters = 2
...

This output reflects the fact that there were two VI functions in the model above, one for each member of the set `i`, and that each of the variables matched to these functions was shadowed by a parameter variable.

Note that there are two QVI models in the GAMS EMP Library, the models [SIMPLEQVII] and [SIMPLEQVII2] (an expanded version of the example shown above).

4.51.4.3 EMP Syntax for Quasi-Variational Inequalities

The general syntax of the EMP annotations used to specify quasi-variational inequalities is as follows:

QVI {0 var [ parameterVar] | [-] equ var [ parameterVar] } { [-] equ}

The EMP keyword `QVI` indicates that this is a quasi-variational inequality specification. All variables and equations included in a QVI must be listed explicitly. First we have the VI functions, their matching variables, and (optionally) the parameter variables shadowing these variables. Note that in a QVI there are no preceding variables as we have in a VI spec:: instead, the implied match to the zero function is indicated by the digit 0 appearing where an equation symbol would otherwise appear. After the function/variable pairs have been listed, the trailing equations (i.e. the equations/constraints defining the mapping `K(·)`) appear.

4.51.5 Equilibrium Problems

While optimization problems have one decision maker that controls all decision variables, equilibrium problems are a collection of optimization problems and variational inequalities, each controlled by a different agent. We typically assume that each variable and each equation is controlled by or belongs to exactly one agent. Variables that are controlled by one agent but appear in the equations of a second agent are regarded as fixed or exogenous variables by that second agent: when taking first-order conditions, the second agent won’t take derivatives wrt these exogenous variables. Later we will relax this assumption and introduce equilibrium problems with shared constraints and shared variables. Note that in this section we will discuss equilibrium problems of the Nash type, i.e. where all agents are on the same level, each assuming the decisions or strategies of the other agents are known and fixed. Equilibrium problems of the Stackelberg type, where there are leaders and followers, are covered in section Bilevel Programs.

We start with the mathematical formulation of an equilibrium problem, next present two examples, and conclude with a description of the EMP annotations for equilibrium problems.
4.51.5.1 Equilibrium Problems: Mathematical Formulation

Consider the following equilibrium problem with \( N \) agents solving minimization problems and one agent solving a variational inequality:

\[
\text{Find } (x^*_1, \ldots, x^*_N, p^*) \text{ satisfying } \begin{align*}
x^*_i &\in \arg \min_{x_i} f_i(x_i, x^*_{-i}) \\
s.t. & \quad g_i(x_i, x^*_{-i}) \leq 0, \text{ for } i = 1, \ldots, N, \\
p^* &\in \text{SOL } (H(p, x^*), K(x^*)), \\
\end{align*}
\]

where \( K(x^*) = \{p \mid w(p, x^*) \leq 0\} \).

Note that \( f_i(x_i, x_{-i}) \) denotes the objective function of the problem of agent \( i \), \( g_i(x_i, x_{-i}) \) are the constraints relating to this optimization problem and \( x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N) \) denotes the decisions of the other agents. Further, \( \text{SOL } (H, K) \) represents the solution set of the variational inequality \( VI(H, K) \).

This problem can be implemented with EMP as follows:

```plaintext
Set i;
Variables obj(i), x(i), p;
Equations deff(i), defg(i), defH, defw;

*Definitions of equations are omitted

Model equil / deff, defg, defH, defw /;

File myinfo /'emp.info'/;
put myinfo 'equilibrium';
loop(i,
   put / 'min', obj(i), x(i), deff(i), defg(i);
);
put 'vi defH p defw' /;
putclose myinfo;

solve equil using EMP;
```

Note that the GAMS variable \( \text{obj}(i) \) holds the value of \( f_i(x) \), \( x(i) \) represents the variable \( x_i \) and \( p \) denotes the variable \( p \), of course. The equations \( \text{deff}(i) \) and \( \text{defg}(i) \) are closed-form definitions of the objective function \( f_i(x) \) and the constraint \( g_i(x) \) respectively. The equation \( \text{defH} \) defines the variational inequality function \( H \) and the equation \( \text{defw} \) defines the set \( K \). The EMP annotations found in the file emp.info specify the equilibrium structure of the model:

```plaintext
equilibrium
min obj('1') x('1') deff('1') defg('1')
...
min obj('N') x('N') deff('N') defg('N')
vi defH p defw
```

The EMP keyword \texttt{equilibrium} indicates that the annotations are for an equilibrium problem. Each of the EMP keywords \texttt{min} begins a new optimization problem (owned by a new agent), where each problem has its own objective variable, decision variables, and equations / constraints. For example, the first agent minimizes \texttt{obj('1')} controls or owns \texttt{x('1')}, and is subject to the constraints \texttt{deff('1')} and \texttt{defg('1')}.

If other variables like \texttt{x('2')} and \texttt{x('3')} appear in \texttt{deff('1')} and \texttt{defg('1')}, they will be treated as exogenous by the first agent. This specification is consistent with the formulation above in (9).

Each agent’s optimization problem can be easily (re)constructed given the EMP annotations. Following the optimization problems of the \( N \) agents we have the VI specification for agent \( p \). The EMP keyword \texttt{vi} is followed by the equation-variable pair \texttt{defH p} defining the VI function \( H \) and the equation \texttt{defw} that defines the feasible set \( K \).

Note that the short form of the solve statement is used for equilibrium problems. Objective variables (if and when they exist) belong to individual agents, not to the model as a whole.
4.51.5.2 Equilibrium Problems with EMP: A Simple Example

Consider the following example from Kim & Ferris (2017) [109]. In this economic equilibrium problem there are three agents: one profit-maximizing producer, one utility-maximizing consumer and a market that determines the price of three commodities based on production and demand. The problem data include a technology matrix \( A \), where the entry \( a_{ij} > 0 \) denotes the output of the commodity \( i \) for each unit of activity of producer \( j \) and \( a_{ij} < 0 \) denotes the respective input. Further, an initial endowment \( b \) and the demand function \( d(p) \) is given, where \( p \) is the price. The consumer maximizes her utility within her budget, which depends on the price \( p \) and the initial endowment \( b \). Let \( y \) represent the activity of the producer, \( x \) represent the demand of the consumer and \( p \) represent the prices of commodities. Then \((y^*, x^*, p^*)\) is a general equilibrium if it satisfies the following:

\[
\begin{align*}
-A^T p^* &\geq 0 \quad \text{No positive profit for each activity} \\
b + A y^* - d(p^*) &\geq 0 \quad \text{No excess demand} \\
p^* \geq 0, y^* &\geq 0 \quad \text{Nonnegativity} \\
-A^T p^* &\perp y^* \quad \text{No activity for earning negative profit and positive activity implies balanced profit} \\
b + A y^* - d(p^*) &\perp p^* \quad \text{Zero price for excess supply and market clearance for positive price}
\end{align*}
\]

The code for the respective model is given below. Note that instead of using the consumer demand function \( d(p) \) in its explicit form, we introduce a utility-maximizing consumer with demand \( x \).

```plaintext
set i 'commodities' /1*3/;
variable u 'consumer utility'; positive variables
   y 'activity of the producer'
x(i) 'Marshallian demand of the consumer'
p(i) 'prices';
parameters
   A(i) 'technology matrix' /1 1, 2 -1, 3 -1/
s(i) 'budget share' /1 0.9, 2 0.1, 3 0/
b(i) 'endowment' /1 0, 2 5, 3 3/;
equations
   profit 'profit of activity'
mkt(i) 'constraint on excess demand'
   udef 'Cobb-Douglas utility function'
   budget 'budget constraint';
   profit.. -sum(i, A(i)*p(i)) =g= 0;
mkt(i).. b(i) + A(i)*y - x(i) =g= 0;
udef.. u =e= sum(i, s(i)*log(x(i)));
budget.. sum(i, p(i)*x(i)) =l= sum(i, p(i)*b(i));
model m /mkt, profit, udef, budget/;
file empinfo '%emp.info%'; putclose empinfo
   'equilibrium' /
      ' max', u, ' x', udef, budget /
      ' vi profit y' /
      ' vi mkt p' /
```

* the second commodity is used as a numeraire
p.fx('2') = 1;
x.l(i) = 1;
solve m using EMP;

Observe that in the EMP annotations the problems of the three agents are specified after the EMP keyword equilibrium: the consumer solves a maximization problem (where the utility \( u \) is maximized) and the activities of the producer and the price-setting market are expressed as VI. As there are three commodities, the first VI actually generates three VI functions, one for each commodity. Thus there are three agents and four VI functions in the equilibrium problem. This is reflected in the EMP Summary in the listing file:

--- EMP Summary
...  
VI Functions = 4
Equilibrium Agent = 3
...  

In the GAMS EMP Library there are several models that have a similar form, e.g. Scarf's activity analysis model [SCARFEMP-DEM] and the simple equilibrium problem [SIMPEQUIL]. The latter demonstrates that there are equilibrium problems where the optimization problems of the individual agents are solvable, but the overall equilibrium problem does not have a solution.

4.5.1.5.3 Equilibrium Problems with EMP: Example with Dual Variables

In many applications equilibrium problems come with a twist: the dual variable associated with a constraint in the problem of one agent appears exogenously in the problem of another agent. The following simple example with two agents is from model [DUALVAR] in the GAMS EMP Library.

Problem of the first agent: Min \( z = v + w \)
\begin{align*}
\text{s.t.} & \quad \sqrt{v + 1} + 2w \geq 2 \quad (\bot \quad u \geq 0) \\
& \quad v, w \geq 0
\end{align*}

Problem of the second agent: VI: 
\( F(y) := y - 4u + 1 \quad (\bot \quad y \text{ free}) \)

N.B.: the variable \( u \) that appears in the problem of the second agent is the dual multiplier (aka shadow price) of the first agent's constraint. This equilibrium problem can be modeled in GAMS with EMP as follows:

positive variables
\( v \) 'belongs to min agent'  
\( w \) 'belongs to min agent'  
\( u \) 'dual of min constraint'  
;  
free variables
\( y \) 'belongs to VI agent'  
\( z \) 'objective var'  
;  
equations
\text{defz} 'objective def'
\text{g} 'constraint for min agent'
The EMP info file contains the EMP keyword equilibrium followed by the specifications for the two agents: a minimization problem for the first agent and a VI for the second agent. The special relationship between the variable u and the equation g is declared via the EMP keyword dualvar followed by the respective variable-equation pair. Recall our usual assumption that each variable and each equation is owned or controlled by exactly one agent. Since variable u is tied to equation g and g is owned by the first agent, variable u is owned by the first agent also.

Besides the number of equilibrium agents and the number of VI functions, the EMP Summary lists the number of dual variable maps:

--- EMP Summary

... Dual Variable Maps = 1 ...

... VI Functions = 1
Equilibrium Agent = 2 ...

Note

Although the example above contained an optimizing agent and a VI agent, dual variables most often occur in equilibrium problems with several optimizing agents.

Other examples with dual variables in the GAMS EMP Library include a formulation of the well-known transportation model as an equilibrium problem [TRANSEQL], Scarf's activity analysis model [SCARFEMP-PRIMAL] and the general equilibrium model [TWO3EMP].
4.51.5.4 EMP Syntax for Equilibrium Problems

The EMP framework provides the following general syntax to specify equilibrium problems:

\[
\begin{align*}
\text{Equilibrium} & \quad \{\text{VIsol \{equ\}}
\{\text{Implicit \{var \text{equ}\}}
\{\text{MAX|MIN \text{obj \{var\} \{-\} \text{equ}}
\{\text{VI \{var\} \{-\} \text{equ var} \{-\} \text{equ}}
\{\text{DualVar \{var [-] \text{equ}\}}
\end{align*}
\]

The EMP keyword \text{Equilibrium} indicates that the specifications that follow define the structure of an equilibrium problem. The \text{MAX}, \text{MIN}, and \text{VI} keywords specify agents in the problem, while the rest of the keywords are optional modifiers used to adjust the structure of the agent models or the meaning of the equations they contain.

\begin{itemize}
  \item \text{Note}: An equilibrium problem must contain at least one agent, i.e. must contain one of the keywords \text{MAX}, \text{MIN}, or \text{VI}.
\end{itemize}

The EMP keyword \text{VIsol} identifies a shared constraint(s) and specifies the MCP reformulation to use for it: see section \text{Equilibrium Problems with Shared Constraints} below for details. The keyword \text{Implicit} identifies a shared variable and its defining constraint: see section \text{Equilibrium Problems with Shared Variables} below for details. The keywords \text{MAX} and \text{MIN} each begin the specification of an optimization agent and are followed by the objective variable \text{obj} and the other variables and equations owned by the agent, as described in the formulation and example above. The keyword \text{VI} begins the specification of a VI agent: see the section on \text{VI} above for details. Finally, the EMP keyword \text{DualVar} specifies that the variable \text{var} is the Lagrange multiplier for the equation \text{equ}. For examples, see sections \text{Equilibrium Problems with EMP: Example with Dual Variables} and \text{Embedded Complementarity Systems}.

The symbol '∗' specifies that the default or automatic assignment of variables to this agent be used, i.e. the set of variables used in the equations owned by this agent but not explicitly or otherwise assigned to another agent. Note that if a variable occurs in equations owned by multiple agents and is not explicitly assigned to any agent, the default assignment is not well defined and using it will be flagged as an error. To avoid confusion and promote clarity, we recommend that modelers use explicit variable lists and avoid the '∗' symbol.

The "-" sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that \(x^{1.5} = L = y\) becomes \(y = G = x^{1.5}\). Flipped equations in EMP behave in the same way as flipped equations in MCP.

\begin{itemize}
  \item \text{Note}: When the EMP keyword \text{equilibrium} appears in the EMP annotations, the solve statement takes the short form also used for complementarity problems in GAMS.
\end{itemize}
4.51.6 Embedded Complementarity Systems

Embedded complementarity systems of the following form arise frequently in applications:

$$
\begin{align*}
\text{Min}_x & \quad f(x, y) \\
\text{s.t.} & \quad g(x, y) \leq 0 \quad (\perp \lambda \geq 0) \\
H(x, y, \lambda) & = 0 \quad (\perp y \text{ free})
\end{align*}
$$

(12)

Note that the optimization problem is over the variable $x$ and it is parametrized by the variable $y$. The choice of $y$ is determined by the complementarity relationships represented here by $H$.

From an EMP perspective, there are two ways to annotate a GAMS model to specify the model above: we will describe both below. These approaches provide equivalent additional information that prompts the EMP tool to automatically create the following MCP:

$$
\begin{align*}
0 & = \nabla_x L(x, y, \lambda) \quad \perp x \text{ free} \\
0 & \leq -\nabla_\lambda L(x, y, \lambda) \quad \perp \lambda \geq 0 \\
0 & = H(x, y, \lambda) \quad \perp y \text{ free},
\end{align*}
$$

(13)

where the Lagrangian is defined as

$$L(x, y, \lambda) = f(x, y) + \lambda^T g(x, y).$$

(4.24)

The first approach uses the EMP keywords `dualequ` and `dualvar`, as contained in the model [FERRIS43].

```plaintext
variables obj, x, y;
positive variable lambda;

equations defobj, g, H;

* We omit the equation definitions here.

model ecs /defobj, g, H/;

file empinfo / '%emp.info%'/; putclose empinfo
'dualequ H y' /
'dualvar lambda g' /;

solve ecs using EMP minimizing obj;
```

The external constraint $H$ is expressed as a standard GAMS equation. The long form of the solve statement is used here, which implies the existence of a single optimizing agent that by default owns all equations and variables. The first EMP keyword `dualequ` indicates that the equation $H$ and the variable $y$ do not belong to the optimizing agent: instead, $y$ is treated as an exogenous variable by this agent, and this agent is assumed to know nothing about the functional form of $H$, so that $H$ will not appear in any first-order conditions. Instead, a complementarity relationship between the function defined by $H$ and the variable $y$ is required to exist at optimality. The EMP keyword `dualvar` indicates that the variable $\lambda$ is the dual of the equation $g$. As a result $\lambda$ will be treated exogenously wherever it appears. Given the EMP annotations for this model, JAMS will automatically reformulate the problem as the MCP in (12) and pass this model to an MCP subsolver.

The EMP Summary produced by JAMS contains the following lines:
The second modeling approach recasts the problem above as an equilibrium problem with two agents: the first agent solves a minimization problem and the second agent solves a VI. The algebra in the model remains the same: only the EMP annotations and the solve statement change.

```
putclose empinfo
'equilibrium' /
' min obj x g defobj' /
' vi H y' /
' dualvar lambda g' /

solve ecs using EMP;
```

The EMP keyword **equilibrium** indicates we have an equilibrium problem. The EMP keyword **min** indicates that the first agent solves a minimization problem with the objective variable **obj**, the decision variable **x** and the equations **g** and **defobj**. In contrast to the first approach, where the default (because of the long-form solve statement) is one optimizing agent owning all equations, here we specify the first agent’s minimization model explicitly and from the ground up. As a result it doesn’t contain the equation **H** and we do not use the **dualequ** keyword to take **H** out. Instead, the EMP keyword **vi** specifies that the second agent solves a VI defined by **H** matched with the variable **y**. The **dualvar** keyword functions here as it did in the previous example.

The EMP Summary produced by JAMS contains the following lines:

```
--- EMP Summary
... Dual Variable Maps = 1
... Dual Equation Maps = 0
... VI Functions = 1
... Equilibrium Agent = 2
... Dual Variable Maps = 1
Dual Equation Maps = 1

The EMP Summary produced by JAMS contains the following lines:

--- EMP Summary
... Dual Variable Maps = 1
... Dual Equation Maps = 0
... VI Functions = 1
... Equilibrium Agent = 2
... Other examples of embedded complementarity systems in the GAMS EMP Library include the simple equilibrium problem [SIMPEQUIL2], the equilibrium problem formulation of the well-known transportation model [TRANSECS], the PIES energy equilibrium problem [PIES], the pure exchange model [NEGISHI] and the spatial price equilibrium model [HARK-MONOP].

It’s worthwhile highlighting the differences between the **dualvar** and **dualequ** keywords, as the two are easily and frequently confused. The **dualvar** keyword makes reference to a constraint owned by an optimizing agent. Derivatives of this constraint, multiplied by the variable referenced, appear in the first-order optimality conditions for this agent. The **dualvar** keyword allows us to use this variable or multiplier explicitly (and in a sense exogenously) in the model algebra. In contrast, the **dualequ** keyword indicates that, contrary to what the default is, an equation is not owned by any optimizing agent, so no derivatives of this equation will appear in any FOC or in the reformulated model. The two are similar in that if a variable **x** appears with either **dualvar** or **dualequ**, no derivatives w.r.t. **x** will appear in the model reformulation.

<table>
<thead>
<tr>
<th><strong>dualvar x F</strong></th>
<th><strong>dualequ F x</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable symbol appears first</td>
<td>Equation symbol appears first</td>
</tr>
<tr>
<td>F is owned by an optimizing agent</td>
<td>F is a system constraint</td>
</tr>
<tr>
<td>Derivatives of F appear in FOC</td>
<td>No derivatives of F appear in derived model</td>
</tr>
<tr>
<td>Does not change ownership of F</td>
<td>F is taken away from an optimizing agent</td>
</tr>
</tbody>
</table>
4.51 Extended Mathematical Programming (EMP)

Table 1: Differences between dualvar and dualequ

4.51.7 Equilibrium Problems with Shared Constraints

In the equilibrium problems we have discussed so far, each variable and each constraint was owned by a single agent. In this section we relax the assumption that each constraint has to be controlled by a single agent and introduce shared constraints. Shared constraints are constraints that appear in the problems of several agents. They are mainly used to model resources shared among agents. Note that the examples in this section are from Kim & Ferris (2017) [109].

Consider the following example:

\[
\begin{align*}
\text{Find } & (x^*_1, \ldots, x^*_N) \text{ satisfying } \\
& x^*_i \in \arg \min_{x_i} f_i(x_i, x^*_{-i}) \\
& \text{s.t. } g_i(x_i, x^*_{-i}) \leq 0 \\
& h(x_i, x^*_{-i}) \leq 0, \quad \text{for } i = 1, \ldots, N.
\end{align*}
\]

(14)

The constraints \(g_i\) are owned by agent \(i\), while the constraint \(h\) is shared by all agents. This example can be reformulated in two different ways, each with its own EMP annotations and resulting MCP. However, both share the model formulation below:

variables obj(i), x(i);
equations deff(i), defg(i), defh;
model sharedc / deff, defg, defh /;

Observe that the equation \(\text{defh}\) implementing the shared constraint \(h(x) \leq 0\) is not indexed: it exists once, not once per agent. The first way to write the EMP annotations is as follows:

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
    put 'min', obj(i), x(i), deff(i), defg(i), defh; /;
);
putclose empinfo;

Here the equation \(\text{defh}\) appears in the minimization problem of each agent. As the equation \(\text{defh}\) is not indexed by agent and appears in the problem of each agent, it is easy to see that it is a shared constraint. Given these annotations, the EMP framework aggregates the FOC for each agent to create the following MCP:

\[
F(z) = ((F_i(z)^T)_{i=1}^N)^T, \\
F_i(z) = \begin{bmatrix}
\nabla_x f_i(x) - \nabla_y g_i(x) \lambda_i - \nabla_z h(x) \mu_i \\
g_i(x) \\
h(x)
\end{bmatrix}, \\
z = ((z_i^T)_{i=1}^N)^T, \\
z_i = \begin{bmatrix}
x_i \\
\lambda_i \leq 0 \\
\mu_i \leq 0
\end{bmatrix}, \quad \text{for } i = 1, \ldots, N.
\]

(15)

Note that the equation \(h(\cdot)\) is replicated for each agent and each agent is assigned a separate multiplier \(\mu_i\).
Attention

By default, the solver JAMS does not allow shared constraints. The option SharedEqu needs to be specified in the option file jams.opt if shared constraints are intended, otherwise an error is reported when they are detected. This cautious default helps minimize the number of surprising results.

A full model including the JAMS option file is given below.

The second way to write the EMP annotations uses the EMP keyword VIsol to specify that a variational equilibrium be computed, i.e. a solution where the multipliers \( \mu_i \) are all equal. The VIsol keyword should appear after the EMP keyword equilibrium and before the descriptions of agent models.

\[
\text{put / 'VIsol defh'};
\]

The multipliers to make equal (or, equivalently, the constraints not to replicate) are indicated by the equation name(s) following the keyword VIsol. These constraints appear once in the resulting MCP, not once per agent. The EMP tool creates the following MCP:

\[
\begin{align*}
F(z) &= \left( (F_i(z)^T)^{N_{\text{im}}}, F_h(z)^T \right)^T, \\
F_i(z) &= \begin{bmatrix} \nabla z_i \cdot f_i(x) - \nabla x_i g_i(x) \lambda_i - \nabla x_i h(x) \mu \end{bmatrix}, \\
F_h(z) &= \begin{bmatrix} h(x) \end{bmatrix}, \\
z &= \left( (z_i^T)^{N_{\text{im}}}, z_h^T \right)^T, \\
z_i &= \begin{bmatrix} x_i \end{bmatrix}, \quad \lambda_i \leq 0, \\
z_h &= [\mu \leq 0].
\end{align*}
\]

If there are no constraints \( g_i(x) \), then - assuming a constraint qualification - the equilibrium problem with the shared constraint corresponds exactly to the variational inequality \( VI(F,X) \), where the set \( X = \{ x | h(x) \leq 0 \} \) and \( F(x) = (\nabla x_i f_i(x)^T)^{N_{\text{im}}} \). This corresponds in turn to the solution of the first MCP (with the replicated shared constraints) in case that solution has the property that the multipliers \( \mu_i \) are all equal. This is what gives the latter solution its name variational equilibrium. If the equilibrium problem with the shared constraint has a unique solution, both MCP reformulations will have the same solution. Otherwise, the two MCPs that correspond to the two different EMP annotations may have different solutions.

An example for shared constraints is the pollution of a river basin in Haurie & Krawczyk (1997) [96] and Krawczyk & Uryasev (2000) [111], where the total amount of pollutants that may be dumped in a river is restricted. This implies that the environmental constraints of some pollutant-producing agents are shared. This example can be formulated as follows, where \( i \) is the agent index and \( m \) denotes the number of shared constraints.

Find \((x_1^*, x_2^*, x_3^*)\) satisfying

\[
\begin{align*}
\text{arg min}_{x_i} \quad & (c_1 i + c_2 i x_i) x_i - \left( d_1 - d_2 \left( \sum_{j=1, j \neq i}^3 x_j^* + x_i \right) \right) x_i \\
\text{s.t.} \quad & \sum_{j=1, j \neq i}^3 x_j^* u_{jm} e_j x_i + u_{im} e_i x_i \leq K_m, \\
& \text{for } i = 1, 2, 3, \quad m = 1, 2,
\end{align*}
\]

where \((c, d, e, u, K)\) is the problem data.

Here three agents produce some commodities and aim to maximize their profit. The term \((c_1 i + c_2 i x_i) x_i\) denotes the total cost for each agent \( i \) and the term \((d_1 - d_2 (\sum_{j=1, j \neq i}^3 x_j^* + x_i)) x_i\) represents the revenue for each agent. The amount of pollutants each agent dumps in the river is limited by the two shared constraints. This problem can be implemented in GAMS with EMP as follows:
Sets i / 1*3 / 
m / 1*2 / ;
alias(i,j) ;

Variable  obj(i) ;
Positive Variable x(i) ;

Parameters K(m) / 1 100, 2 100 / 
d1 / 3 / 
d2 / 0.01 / 
e(i) / 1 0.5, 2 0.25, 3 0.75 / ;

Table c(m,i)
   1     2     3
1  0.1  0.12  0.15
2  0.01 0.05  0.01 ;

Table u(i,m)
   1     2
1  6.5  4.583
2  5.0  6.250
3  5.5  3.750 ;

Equations objdef(i)
       cons(m) ;

objdef(i)..  obj(i)  =e=  (c('1',i) + c('2',i)*x(i))*x(i) - (d1 - d2*sum(j, x(j)))*x(i);
cons(m)..  sum(i, u(i,m)*e(i)*x(i)) =l= K(m);

Model m_shared / objdef, cons /;

File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'VIsol cons' /;
loop(i,
    put 'min', obj(i), x(i), objdef(i), 'cons' /;
);
putclose empinfo;

$echo SharedEqu > jams.opt
m_shared.optfile = 1;

solve m_shared using emp;
display x.l, cons.m;

Note that the shared environmental constraints are expressed with the equation cons. In the EMP annotations, we have chosen the formulation with the EMP keyword VIsol. It leads to the solution $x^* = (21.145, 16.028, 2.726)$. Alternatively, we could delete the line with the EMP keyword VIsol and thus prompt the framework to compute the MCP in (15), resulting in the solution $x^* = (0, 6.473, 22.281)$.

### 4.51.8 Equilibrium Problems with Shared Variables

In the last section, we relaxed the restriction that each constraint has to be controlled by a single agent and introduced shared constraints. In this section, we go a step further and allow shared variables. Note that the content of this section is adapted from Kim & Ferris (2017) [109].
First, we introduce the notion of *implicit variables*. In mathematical terms, a variable \( y \) is called an implicit variable if for each value of \( x \) there is at most one value of \( y \) satisfying \((y, x) \in X\). For such implicit variables there exists one and only one function \( g(\cdot) \) such that \((g(x), x) \in X\), where \( g \) is defined over the set \( \{ x \mid \exists y \text{ such that } (y, x) \in X \} \). The set \( X \) is called the *defining constraint* of the variable \( y \): the value of \( y \) is implicitly defined by the value of \( x \) via the defining constraint. In the current implementation, the defining constraint may only be represented as a system of equations and implicit variables must be free variables in GAMS.

*Shared variables* in equilibrium problems are implicit variables that have the same defining constraint for all agents that share the variable. Hence, the defining constraint becomes a shared constraint. Consider the following equilibrium problem with the shared decision variable \( y \) and its defining constraint \( X = \{ (y, x) \mid H(y, x) = 0 \} \):

$$
\begin{align*}
\text{Find} & \quad (y^*, x^*_1, \ldots, x^*_N) \\
\text{arg min} & \quad f_i(y, x_i, x^*_{-i}) \\
\text{s.t.} & \quad H(y, x_i, x^*_{-i}) = 0, \quad \text{for } i = 1, \ldots, N,
\end{align*}
\quad (18)
$$

Note that there are \( N \) agents, \( m \) is the dimension of the variable \( y \) and \( n = \sum n_i \), where \( n_i \) is the dimension of \( x_i \). Assuming we have 4 agents, this example can be implemented in GAMS EMP as follows:

```gams
Set      i / 1*4/;
Variables obj(i), x(i), y;
Equations deff(i), defH;
Model    sharedv / deff, defH /;
File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'implicit y defH' /;
loop (i,
     put 'min', obj(i), x(i), y, deff(i) /;
);
putclose empinfo;
solve sharedv using EMP;
```

In the EMP annotations, the EMP keyword *implicit* is used to declare the implicit variable \( y \) and its defining constraint \( \text{defH} \). Note that the keyword *implicit* must be followed by variable-constraint pairs. If multiple pairs are specified with a single keyword *implicit*, they will be augmented to form a single vector of implicit variables and its defining constraint.

Note

Implicit variables are declared *before* the agent problems are defined.

Observe that the shared variable \( y \) appears in the problem specification for each agent. However, the defining equation \( \text{defH} \) does not appear in each problem specification, since it is assumed to be part of the implicit variable.
4.51.8.1 Reformulation Strategies for Equilibrium Problems with Shared Variables

Like other equilibrium problems, equilibrium problems with shared variables are reformulated as MCPs by the EMP tool. Users can choose between three different reformulation strategies if shared decision variables are involved. In the first strategy, the shared variables are replicated for each agent and the respective KKT conditions are computed, resulting in the following MCP:

\[
F(z) = [(F_i(z)^T)^N_{i=1}, F_h(z)^T]^T, \quad z = [(x_i^T)^N_{i=1}, z_h^T]^T
\]

\[
F_i(z) = \left[\begin{array}{c}
\nabla_x f_i(x, y) - \left(\nabla_x, H(y, x)\right) \mu_i \\
\nabla_y f_i(x, y) - \left(\nabla_y, H(y, x)\right) \mu_i \\
H(y, x)
\end{array}\right], \quad z_i = \begin{bmatrix} x_i \\ \mu_i \end{bmatrix}, \quad z_h = [y].
\]

(19)

Note that in this MCP the constraint \( H \) and the variable \( y \) are replicated \( N \) times. The size of the MCP is \((n + 2mN)\). This reformulation is obtained by specifying the option \( \text{ImplVarModel}=\text{replication} \) in the JAMS options file.

The second reformulation strategy involves switching each shared variable with the multiplier associated with its defining equation. This technique can be applied if all of the following three conditions are met:

1. The defining constraint is given as an equation.
2. The dimension of the range of the defining constraint equals the dimension of the shared variable.
3. The shared variable is a free (unbounded) variable.

The switching strategy uses the fact that in an MCP, the matching between free variables and equations is somewhat arbitrary: it can be re-assigned without changing the solution. Applying this technique, we obtain the following MCP:

\[
F(z) = [(F_i(z)^T)^N_{i=1}, F_h(z)^T]^T, \quad z = [(x_i^T)^N_{i=1}, z_h^T]^T
\]

\[
F_i(z) = \left[\begin{array}{c}
\nabla_x f_i(x, y) - \left(\nabla_x, H(y, x)\right) \mu_i \\
\nabla_y f_i(x, y) - \left(\nabla_y, H(y, x)\right) \mu_i \\
H(y, x)
\end{array}\right], \quad z_i = \begin{bmatrix} x_i \\ \mu_i \end{bmatrix}, \quad z_h = [y].
\]

(20)

Observe that in this MCP, the defining constraint \( H \) and the shared variable \( y \) appear only once. Thus the size of the problem is reduced to \((n + mN + n)\). The EMP framework will use this reformulation if the option \( \text{ImplVarModel}=\text{switching} \) is specified in the JAMS option file. This is currently the default strategy used.

The third strategy is selected by specifying option \( \text{ImplVarModel}=\text{substitution} \) in the JAMS option file. It uses the implicit function theorem to substitute the multipliers \( \mu_i \) by new variables \( \Lambda_i \), where \( \Lambda_i = \nabla_x, H(\nabla_y H)^{-1} \). This technique may be applied when the three conditions of the switching strategy are satisfied, and in addition, the implicit function theorem holds for the defining constraints. The basic idea is to regard the shared variable \( y \) as a function of other non-shared variables and apply the derivative. At each solution to the problem \((y^*, x^*)\), there exists a locally defined implicit function \( h_{x^*}(x) \) such that \( y^* = h_{x^*}(x^*) \) and \( H(h_{x^*}(x), x) = 0 \) for each \( x \) in some neighborhood of \( x^* \). It can be shown that given the implications of the implicit function theorem and the new variables \( \Lambda_i \), the equilibrium problem can be reformulated as the MCP that follows. However, we omit a step-by-step mathematical derivation here, it is given in Kim & Ferris (2017) [109].

\[
F(z) = [(F_i(z)^T)^N_{i=1}, F_h(z)^T]^T, \quad z = [(x_i^T)^N_{i=1}, z_h^T]^T
\]

\[
F_i(z) = \left[\begin{array}{c}
\nabla_x f_i(x, y) - \left(\nabla_y, f_i(x, y)\right) \Lambda_i \\
\nabla_y H(y, x) \Lambda_i - \left(\nabla_x, H(y, x)\right) \Lambda_i \\
H(y, x)
\end{array}\right], \quad z_i = \begin{bmatrix} x_i \\ \Lambda_i \end{bmatrix}, \quad z_h = [y].
\]

(21)
The size of this MCP is \( (n + mn + m) \). It can be significantly reduced if the shared variable is *explicitly* defined, for example, \( H(y, x) = y - h(x) \). In this case, \((\nabla_y H)^{-1}\) is the identity matrix, therefore it is not necessary to introduce the variables \( \Lambda_i \) and the MCP takes the following form:

\[
F(z) = \left[ (F_i(z))_{i=1}^N, F_h(z) \right]^T, \\
F_i(z) = \left[ \nabla x_i f_i(x, y) - \nabla x_i H(y, x) \nabla y f_i(x, y) \right], \\
F_h(z) = \left[ H(y, x) \right], \\
z = \left[ z_i^T \right]_{i=1}^N, z_h^T
\]  

(22)

The size of this MCP is \( (n + m) \), which is a huge decrease compared to the other formulations. Note that the EMP framework detects automatically if the shared variable is explicitly defined and exploits this fact. In the following table an overview of the size of the MCP for the different reformulation techniques is given.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Size of the MCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>((n + 2mN))</td>
</tr>
<tr>
<td>Switching</td>
<td>((n + mN + m))</td>
</tr>
<tr>
<td>Substitution (implicit)</td>
<td>((n + mn + m))</td>
</tr>
<tr>
<td>Substitution (explicit)</td>
<td>((n + m))</td>
</tr>
</tbody>
</table>

Table 2: Size of the Reformulated MCP For Different Reformulation Techniques

### 4.51.8.2 Equilibrium Problems with Shared Variables: A Simple Example

The following simple example computes a saddle point of the Lagrangian associated with a minimization over \( x \) subject to a single equality constraint with dual multiplier \( y \). The GAMS EMP implementation is from Youngdae Kim. The primal (minimizing) and dual (maximizing) agents share the variable \( L \) containing the value of the Lagrangian, as well as its defining equation \( \text{defL} \):

```gams
set i / 1*2 /;
variables
   L  'Lagrangian function: f(x) - y * h(x)'
   x(i) 'primal variables'
   y  'dual variable'
;
equation defL;
defL..  
   L =e= sum{i, sqr(x(i)-1)} - y*(sum{i, x(i)} - 4);
model m / defL /;
file empinfo / '%emp.info%' /; putclose empinfo
   'equilibrium' /  
   'implicit L defL' /  
   ' min L x' /  
   ' max L y' /  
;
solve m using emp;
```

Note that the variable \( L \) is not indexed and it appears in the optimization problem of each agent. It is declared to be an implicit variable (using the keyword *implicit*) in the first line following the *equilibrium* keyword. Its defining constraint \( \text{defL} \) is listed only once, in this same line: it does not appear in the problem specification of each agent. By modeling the Lagrangian as a shared variable, its defining equation does not have to be replicated for each agent.

For other, more complex examples, both with shared decision variables and with a shared objective variable, see Kim & Ferris 2017 [109].
4.51.9 Bilevel Programs

Bilevel programs are mathematical programs with optimization problems in their constraints. The main problem is called the upper-level problem or the leader and the nested problem is called the lower-level problem or the follower. A simple example is the bilevel programming problem that optimizes an upper-level objective over constraints that include a lower-level optimization problem. A famous example from economics is the Stackelberg game, where there is one leader and many followers. Bilevel programming is used in many areas, for example the design of optimal tax instruments: the tax instrument is modeled in the upper level and the clearing market is modeled in the lower level.

In this section we first present a mathematical formulation of a bilevel program with one follower and introduce the respective EMP annotations. Then we present three examples: a simple first example, an example with a variational inequality as the lower-level problem and an example with multiple followers that form a Nash equilibrium. We conclude the section with a short discussion of the general EMP syntax for bilevel programs.

4.51.9.1 Bilevel Programs: Mathematical Formulation

A bilevel program with one leader and one follower can be expressed as follows:

\[
\begin{align*}
\text{Min}_{x \in X, y} & \quad f(x, y) \\
\text{s.t.} & \quad h(x, y) \leq 0 \\
& \quad y \text{ solves } \text{Min}_{y} g(x, y) \\
& \quad \text{s.t. } k(x, y) \leq 0,
\end{align*}
\]

(23)

where

- \( x \in \mathbb{R}^n \) are the upper-level variables,
- \( y \in \mathbb{R}^m \) are the lower-level variables,
- \( f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R} \) is the upper-level objective function,
- \( g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R} \) is the lower-level objective function,
- \( h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^u \) are the upper-level constraints, and
- \( k : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^l \) are the lower-level constraints.

Note that the upper-level constraints do not bind the lower-level decision maker.

This problem can be implemented with EMP as follows:

Sets i, j ;
Variables x(i), objout, y(i), objin;
Equations deff(x,y), defg(x,y), defh(x,y), defk(x,y);

* Definitions of equations are omitted

Model bilevel / deff, defg, defh, defw /;

$onecho > "%emp.info%"
bilevel x
min objin y defg defk
$offecho

solve bilevel using EMP minimizing objout ;
Note that the variables and equations of the program are defined in GAMS in the usual way. The special bilevel structure of the model is specified in the EMP annotations. The EMP keyword **bilevel** indicates that this is a bilevel problem. The name and direction of the leader’s objective variable is taken from the solve statement, but the other variables owned by the leader are listed after the **bilevel** keyword. The lower-level problem is specified next via the EMP keyword **min**, followed by the follower’s objective variable **objin**. The other variables and the equations of the lower-level problem are listed next. Observe that this is exactly the same syntax that we introduced **above** for specifying optimization problems in the context of equilibrium problems.

The EMP tool reformulates the bilevel problem as a Mathematical Program with Equilibrium Constraints (MPEC) and passes this on to a subsolver (e.g. NLPEC or KNITRO). The reformulation is obtained by replacing the lower-level optimization problem by its KKT conditions, resulting in the following problem:

\[
\begin{align*}
\text{Min}_{x \in X, y, \lambda} & \quad f(x, y) \\
\text{s.t.} & \quad h(x, y) \leq 0 \\
& \quad k(x, y) \leq 0 \\
& \quad \lambda_i \leq 0, \quad i = 1, \ldots, l \\
& \quad \lambda_i k_i(x, y) = 0, \quad i = 1, \ldots, l \\
& \quad \nabla_y \mathcal{L}(x, y, \lambda) = 0,
\end{align*}
\]

where

\[
\mathcal{L}(x, y, \lambda) = g(x, y) + \sum_{i=1}^{l} \lambda_i k_i(x, y)
\]

is the Lagrangian function associated with the lower-level problem.

Note, however, that this reformulation is potentially problematic. KKT conditions require theoretical assumptions (like convexity) to be necessary and sufficient for local optimality. There may be cases where the lower level problem has multiple local solutions, but the modeler is interested in the global solution. Hence the reformulation above may not lead to the global solution, even if a global subsolver is used within the solver NLPEC.

Observe that there are two variations to problem (23) that the EMP framework is equipped to handle: the lower-level problem may be a variational inequality instead of an optimization problem, and there may be multiple lower-level problems (optimizing and/or VI) each behaving in a Nash manner. Examples follow in sections Bilevel Programs with EMP: A VI as Follower and Bilevel Programs with EMP: Multiple Followers respectively.

### 4.51.9.2 Bilevel Programs with EMP: A Simple Example

To illustrate, we use model [BARD851](BARD851) from the GAMS EMP Library. Mathematically, the problem is

\[
\begin{align*}
\text{Min}_{x, y_1, y_2} & \quad (x - 1)^2 + 2y_1^2 - 2x \\
\text{s.t.} & \quad (y_1, y_2) \quad \text{solve} \quad \text{Min}_{y_1, y_2} \quad (2y_1 - 4)^2 + (2y_2 - 1)^2 + xy_1 \\
& \quad \text{s.t.} \quad 4x + 5y_1 + 4x_2 \leq 12 \\
& \quad -4x + 5y_1 + 4x_2 \leq -4 \\
& \quad 4x - 4y_1 + 5x_2 \leq 4 \\
& \quad -4x + 4y_1 + 4x_2 \leq -4 \\
& \quad y_1, y_2 \geq 0
\end{align*}
\]

Note that this problem does not have any upper-level constraints. The GAMS code follows.
Positive Variables x, y1, y2;
Variables         objout, objin;

Equations defout, defin, e1, e2, e3, e4;

\[
defout.. \quad \text{objout} = e= \text{sqr}(x-1) + 2*\text{sqr}(y1) - 2*x; \\
defin.. \quad \text{objin} = e= \text{sqr}(2*y1-4) + \text{sqr}(2*y2-1) + x*y1;
\]
\[
e1.. \quad 4*x + 5*y1 + 4*y2 =l= 12; \\
e2.. \quad -4*x - 5*y1 + 4*y2 =l= -4; \\
e3.. \quad 4*x - 4*y1 + 5*y2 =l= 4; \\
e4.. \quad -4*x + 4*y1 + 5*y2 =l= 4;
\]

Model bard / all /;

$echo bilevel x min objin y1 y2 defin e1 e2 e3 e4 > "%emp.info%"

solve bard use emp min objout;

The leader minimizes the variable objout, as specified in the solve statement. The EMP annotations specify the rest of the bilevel structure: the variable x belongs to the upper-level problem, and there is one lower-level agent or problem with objin minimized over the variables y1 and y2 subject to the constraints or equations defin, e1, e2, e3 and e4.

Alternatively, we could write the EMP annotations as:

$echo bilevel x min objin * defin e1 e2 e3 e4 > "%emp.info%"

Here the lower-level variables are not listed: instead, the '∗' indicates that all variables in the GAMS model not explicitly assigned to any agent are to be assigned to the follower.

The EMP Summary produced by JAMS gives the number of followers in the bilevel program:

--- EMP Summary
   Bilevel Followers = 1

The GAMS EMP Library contains other bilevel examples, including several models bard∗, the engineering models [CCMG74] and [CCMG153], and the simple nonconvex model [MIRRLESS]. The models [JOINTC1] and [JOINTC2] illustrate the interplay of the decision variables of the upper-level and lower-level problems.

4.51.9.3 Bilevel Programs with EMP: A VI as Follower

If the lower-level problem is a variational inequality, problem (23) will take the following form:

\[
\begin{align*}
\text{Min}_{x,y} & \quad f(x,y) \\
\text{s.t.} & \quad h(x,y) \leq 0 \\
& \quad y \text{ solves } VI(F, K(x)),
\end{align*}
\] (26)

where
\( x \in \mathbb{R}^n \) are upper-level variables and \( y \in \mathbb{R}^m \) are lower-level variables, and
\( K(x) \subseteq \mathbb{R}^m \) is a closed convex set, parametrized by \( x \).

Consider the following simple example adapted from [MULTMPEC]:

\[
\begin{align*}
\text{Min}_{w,z} & \quad z \\
\text{s.t.} & \quad e^z + w = 2 \\
& \quad z \geq 1 \\
& \quad w \text{ solves the } VI(F, K), \text{ where } K = \mathbb{R} \text{ and } F = w + z + 3
\end{align*}
\]

This bilevel problem can be modeled with EMP as follows:

variables w, z;
equations h, F;
h.. exp(z) + w =e= 2;
F.. w + z =n= -3;
z.lo = 1;
model bpvi / h, F /;

solve bpvi using emp min z;

Note that in the EMP annotations the lower-level problem is specified as a VI with the VI function \( F \) and the corresponding variable \( w \): for details see section EMP Syntax for Variational Inequalities. The count of VI functions is indicated in the EMP Summary:

--- EMP Summary
...
VI Functions \quad = 1
...  
Bilevel Followers \quad = 1

4.51.9.4 Bilevel Programs with EMP: Multiple Followers

The EMP framework allows multiple followers in a bilevel program. Taking the leader decisions as fixed, these followers behave as the agents in a Nash equilibrium. Consider the engineering example [CCMG71] from the GAMS EMP Library. In this bilevel program there are two followers, both solving minimization problems, specified in the EMP annotations as follows:

...
$onecho > %emp.info%
bilevel x1 x2 x3 x4
  min h1 u1 u2 u3 u4 defh1 e1
  min h2 v1 v2 v3 v4 defh2 e2
$offecho
...


The variables $x_1$, $x_2$, $x_3$ and $x_4$ are owned or controlled by the leader. The first follower minimizes the variable $h_1$ over the variables $u_1$, $u_2$, $u_3$ and $u_4$ subject to the constraint/equation $e_1$. The second follower is defined in a similar way. More details on how to specify optimization subproblems in the EMP annotations are given in section EMP Syntax for Equilibrium Problems.

Observe that in the EMP annotations of the actual model [CCMG71] the following shorthand notation is used:

```
... $onecho > %emp.info%
bilevel x1 x2 x3 x4
min h1 * defh1 e1
min h2 * defh2 e2
$offecho ...
```

The symbol '*\  ' in the problem of the first follower indicates that this agent will optimize over all variables appearing in equations $defh1$ and $e1$ (i.e. the equations it controls) that are not claimed by another agent. In this case, the respective variables are $u_1$, $u_2$, $u_3$ and $u_4$. N.B.: in this example, if a variable appears in the equations for both followers the problem structure is not well defined and an error will result. To avoid confusion and promote clarity, we recommend that modelers explicitly list all the variables in each agent's problem, as shown in the first version of the EMP annotations above.

The number of followers is indicated in the EMP Summary:

```
--- EMP Summary
...
Bilevel Followers   = 2
```

The GAMS EMP Library contains several bilevel problems with several followers, e.g. the well-known transportation model with a variable demand function cast as a bilevel problem with one optimization follower and one VI follower [TRANSBP], a simple example with two VI followers [MULTMPEC], and the spatial equilibrium Stackelberg model [HARK-STACK].

### 4.51.9.5 EMP Syntax for Bilevel Programs

The general syntax that EMP provides to specify bilevel programs in the EMP annotations file `emp.info` is as follows:

```
Bilevel {var}
{MAX|MIN obj {var|*} {[-] equ}}
{VI {var|*} {[-] equ var} {[-] equ}}
{Dualvar {var [-] equ}}
```

A bilevel program is declared with the EMP keyword `bilevel`, followed by the decision variables of the upper-level problem. The other specifications refer to the lower-level problems. These lower-level followers are optional, but at least one follower has to be specified. Optimization problems are defined using the EMP keyword `max` or `min` followed by the objective variable `obj` and the decision variables and the equations that implement the constraints of the problem. Variational inequalities are introduced with the EMP keyword `vi`; details of the specification are given in section EMP Syntax for Variational Inequalities.

The `dualvar` keyword is used in the same way as it is for equilibrium problems.

Note

While the EMP framework allows the symbol '*\  ' to be used to specify an automatically-created list of variables, we recommend against using the '*\  ' in the annotations file. Instead, list all variables explicitly to make the structure of the model clear and unambiguous to both human and machine readers of the EMP info file.
4.51.10 Disjunctive Programming

Disjunctive programming is an alternative modeling approach to mixed integer programming. Both mathematical programs model optimization problems that involve discrete and continuous variables. The advantage of disjunctive programming is that it retains and exploits the inherent logic structure of problems and thus reduces the combinatorics and improves the relaxations by using Boolean variables and disjunction definitions for modeling discrete choices.

Disjunctive programs have many applications, including ordering of tasks in a production process, organizing complex projects in a time saving manner and choosing the optimal route in a circuit.

In this section we first present the mathematical formulation of Generalized Disjunctive Programs (GDPs) and then demonstrate how disjunctive programs are implemented with GAMS EMP using two simple examples. Note that both examples are adapted from the LogMIP 2.0 User’s Manual. At the end of the section we introduce and discuss the syntax for EMP annotations for disjunctive programs.

For information on the development of disjunctive programming in GAMS EMP and its connection to the solver LogMIP, see the respective section in the solver manual of the solver JAMS.

4.51.10.1 Generalized Disjunctive Programs (GDPs)

A GDP has Boolean and continuous variables, algebraic constraints that need to be satisfied regardless of the discrete choices, disjunctions that represent the discrete choices, and logic propositions that contain the logic relationships between the Boolean variables. Mathematically, the general structure of a GDP may be expressed as follows:

\[
\begin{align*}
\text{Min} & \quad f(x) \\
\text{s.t.} & \quad g(x) \leq 0 \\
& \quad \bigvee_{i \in D_k} \left( Y_{ik} \cdot r_{ik}(x) \right) \leq 0, \quad k \in K \\
& \quad \Omega(Y) = \text{True} \\
& \quad L \leq x \leq U, \quad x \in \mathbb{R}^n, \\
& \quad Y_{ik} \in \{\text{True}, \text{False}\}
\end{align*}
\]

(28)

where:

- \( f : \mathbb{R}^n \to \mathbb{R} \) is a function, \( x \) is a vector of continuous variables with bounds \( L \) and \( U \).
- \( g : \mathbb{R}^n \to \mathbb{R}^l \) represents the set of global constraints.
- Each disjunction \( k \in K \) is composed of a number of terms \( i \in D_k \) that are connected by the Boolean operator \( \lor \) (\( \lor \)).
- Each term \( i \in D_k \) consists of a Boolean variable \( Y_{ik} \) and a set of inequalities \( r_{ik}(x) \leq 0, \quad r_{ik} : \mathbb{R}^n \to \mathbb{R}^j \).
- If \( Y_{ik} \) is true, then \( r_{ik}(x) \leq 0 \) is enforced, otherwise these constraints are ignored.
- \( \Omega(Y) = \text{True} \) are logic propositions for the Boolean variables \( Y_{ik} \) expressed in the Conjunctive Normal Form \( \Omega(Y) = \bigwedge_{t=1,2,\ldots,T} \left[ \bigvee_{Y_{jk} \in R_t} (Y_{jk}) \cdot \bigvee_{Y_{jk} \in Q_t} (\neg Y_{jk}) \right] \), where for each clause \( t = 1, \ldots, T \), \( R_t \) is the subset of Boolean variables that are non-negated and \( Q_t \) is the subset of Boolean variables that are negated.

N.B.: we assume that each disjunction is an exclusive-or, so that for each \( k \) exactly one variable \( Y_{ik} \) is true. Put another way, we assume the logic constraints \( \bigvee_{i \in D_k} Y_{ik} \) are contained in \( \Omega(Y) = \text{True} \).

There are three cases of disjunctive programs: the functions \( f, g \) and \( r \) are linear, some of them are nonlinear, but convex, and some of them are nonlinear and nonconvex. Note that currently GAMS EMP facilitates modeling only the first two cases.
4.51.10.2 Disjunctive Programming with EMP: Example with No Algebraic Constraints

Consider the following simple example, that has no algebraic constraints which must be satisfied regardless of the disjunctive choices:

\[
\begin{align*}
\text{Minimize} & \quad c + 2x_1 + x_2 \\
\text{subject to} & \quad \begin{bmatrix}
Y_1 \\
-x_1 + x_2 + 2 \leq 0 \\
c \leq 5
\end{bmatrix} \lor \begin{bmatrix}
Y_2 \\
2 - x_2 \leq 0 \\
c \leq 7
\end{bmatrix} \\
\begin{bmatrix}
Y_3 \\
x_1 - x_2 \leq 0
\end{bmatrix} \lor \begin{bmatrix}
\neg Y_3 \\
x_1 \leq 1
\end{bmatrix}
\end{align*}
\]

Disjunctions

\[
\begin{align*}
Y_1 \land \neg Y_2 & \Rightarrow \neg Y_3 \\
Y_2 & \Rightarrow \neg Y_3 \\
Y_3 & \Rightarrow \neg Y_2
\end{align*}
\]

Logic Propositions (29)

Continuous Variables

\[
\begin{align*}
0 & \leq x_1 \leq 5 \\
0 & \leq x_2 \leq 5 \\
c & \geq 0
\end{align*}
\]

Boolean Variables

\[
Y_j \in \{\text{True, False}\}, \quad j = 1, 2, 3
\]

Observe that there are two disjunctions, each with two terms. In the first disjunction, each term is governed by a different Boolean variable: the first term is active if \(Y_1\) is true and the second term is active if \(Y_2\) is true. In the second disjunction, both terms are governed by the Boolean variable \(Y_3\): the first term applies if \(Y_3\) is true and the second term applies if \(Y_3\) is false.

Note that the logic propositions imply that if \(Y_1\) is true and \(Y_2\) is false, then \(Y_3\) must be false, and that \(Y_2\) and \(Y_3\) cannot both be true.

This example can be implemented in GAMS EMP as follows:

\[
\begin{align*}
\text{Set} \quad & i / 1*2 / \\
& j / 1*3 /; \\
\text{Positive Variables} \quad & x(i), c; \\
\text{Variable} \quad & z; \\
\text{Binary Variables} \quad & y(j); \\
x.\text{up}(i) = 5; \\
c.\text{up} = 7; \\
\text{Equations} \quad & \text{Obj, Eq1, Eq2, Eq3, Eq4, Eq5, Eq6}; \\
\text{Obj..} \quad & z =e= c + 2*x('1') + x('2'); \\
\text{* Equations for Disjunctions} \\
\text{Eq1..} \quad & x('2') - x('1') =l= - 2; \\
\text{Eq2..} \quad & c =l= 5; \\
\text{Eq3..} \quad & x('2') =g= 2; \\
\text{Eq4..} \quad & c =l= 7; \\
\text{Eq5..} \quad & x('1') - x('2') =l= 1; \\
\text{Eq6..} \quad & x('1') =l= 1;
\end{align*}
\]
* Equations for Logic Propositions

Logic Equations LEq1, LEq2, LEq3;

LEq1.. y('1') and not y('2') -> not y('3');
LEq2.. y('2') -> not y('3');
LEq3.. y('3') -> not y('2');

Model small1 / all /;

File emp / '%emp.info%' /;
put emp;
$onput
disjunction y('1') Eq1 Eq2 elseif y('2') Eq3 Eq4
disjunction y('3') Eq5 else Eq6
$offput
putclose;

Option optcr = 0.0;

solve small1 using EMP minimize z;

Note that in this model the Boolean variables $Y_j$ are implemented as GAMS binary variables, the inequalities in the terms of the disjunctions are formulated as GAMS equations, and the logic propositions are expressed as GAMS logic equations. The disjunctive structure of the model is specified in the EMP annotations file emp.info. This file contains two lines (one per disjunction), each starting with the EMP keyword disjunction and specifying the structure and content of its disjunction. In the first line or disjunction, the binary variable $y('1')$ that governs the first term is followed by the two equations contained in this term. The EMP keyword elseif denotes the start of a new term, here governed by the binary variable $y('2')$ listed next and containing the two equations Eq3 and Eq4. Similarly, in the second line, the binary variable $y('3')$ that governs the first term of the second disjunction is followed by the equation Eq5 contained in that term. As the binary variable governing the second term is just the negation of $y('3')$, the keyword else is enough to specify this and is followed by the equation Eq6 of the second term.

Note

Much more complex logical constructs for disjunctions are possible. For details, see section EMP Syntax for Disjunctive Programming below.

Finally, note that the model type in the solve statement is EMP.

Given the annotations in the file emp.info, the solver JAMS reformulates the model as a MIP (Mixed Integer Programming) model and passes it to a subsolver. By default, the convex hull relaxation is used for the reformulation, but users may choose reformulations that use big M or indicator constraints: see section EMP Syntax for Disjunctive Programming for details.

Observe that the listing file will contain some additional information if the model type EMP is used. The EMP Summary and the Disjunction Summary may be particularly useful. The respective listings for our example model follow:

--- EMP Summary
  Logical Constraints = 3
  Disjunctions = 2

--- Disjunction Summary
  Disjunction 1 Term 2 is active
  Disjunction 2 Term 2 is active

Note that the EMP summary lists the number of logic constraints and disjunctions and the disjunction summary reports which terms of the disjunctions are active in the optimal solution.
4.51.10.3 Disjunctive Programming with EMP: Example with No Logic Propositions

Consider the following simple example:

\[
\begin{align*}
\text{Min} & \quad t \\
\text{s.t.} & \quad t \geq x_A + 8 \\
& \quad t \geq x_B + 5 \\
& \quad t \geq x_C + 6 \\
& \quad \left[ Y_1 (x_A - x_C + 5) \right. \\ & \quad \left. - Y_1 (x_C - x_A + 2) \leq 0 \right] \\
& \quad \left[ Y_2 (x_B - x_C + 1) \right. \\ & \quad \left. - Y_2 (x_C - x_B + 6) \leq 0 \right] \\
& \quad \left[ Y_3 (x_A - x_B + 5) \right. \\ & \quad \left. - Y_3 (x_B - x_A) \leq 0 \right] \\
& \quad t, x_A, x_B, x_C \geq 0 \\
& \quad Y_j \in \{\text{True, False}\}, j = 1, 2, 3
\end{align*}
\]

As this example has no logic propositions, the binary variables would not appear in any equations of a GAMS formulation like the one above. As a result, the binary variables would not be part of the model and any EMP annotations file that mentions these variables would be rejected by the EMP solver. To avoid this problem, we can introduce a dummy equation and include it in the EMP model: this ensures that the binary variables will be part of the GAMS model. The respective code follows:

```
set i / A, B, C /  
  j / 1*3 /  
  
positive variables x(i), t;  
binary variables y(j);  
variable z;  

equations obj, alg1, alg2, alg3, 
  d1t1, d1t2, d2t1, d2t2, d3t1, d3t2 
  dummy;  

obj.. z =e= t;  

* common algebraic equations  
alg1.. t =g= x('A') + 8;  
alg2.. t =g= x('B') + 5;  
alg3.. t =g= x('C') + 6;  

* equations for disjunctions  
d1t1.. x('A') - x('C') + 5 =l= 0;  
d1t2.. x('C') - x('A') + 2 =l= 0;  
d2t1.. x('B') - x('C') + 1 =l= 0;  
d2t2.. x('C') - x('B') + 6 =l= 0;  
d3t1.. x('A') - x('C') + 5 =l= 0;  
d3t2.. x('B') - x('C') =l= 0;  

* dummy equation  
dummy.. sum(j, y(j)) =g= 0;  
```
model small2 / all /
file emp / '%emp.info%' /

putclose emp
"disjunction y('1') d1t1 else d1t2" /
"disjunction y('2') d2t1 else d2t2" /
"disjunction y('3') d3t1 else d3t2" /
;
option optcr = 0.0;
solve small2 using EMP minimize z;

Apart from the dummy equation, this formulation is very similar to the formulation in the first example above.

EMP also supports an alternative formulation for models that have no logic equations, a formulation using implicit default binary variables. These variables are denoted in the EMP annotations with the star symbol ‘∗’, which is internally replaced by a default binary variable. Note that this alternative model does not contain any explicit binary variables $Y(j)$ and hence the dummy equation may be omitted, as in the following GAMS code which can be added to the model above:

model small3 'no dummy equation needed' / small2 - dummy /

putclose emp
"disjunction * d1t1 else d1t2" /
"disjunction * d2t1 else d2t2" /
"disjunction * d3t1 else d3t2" /
;
solve small3 using EMP minimize z;

4.51.10.4 EMP Syntax for Disjunctive Programming

The general syntax that EMP provides to write disjunctions to the EMP annotations file emp.info is as follows:

\[\text{Disjunction [chull [chull eps] | bigM [big M value] | indic]}
\[\quad \text{[NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]\]

The EMP keyword Disjunction is mandatory, it indicates that what follows is a disjunction. The three constructs that follow are optional and relate to the three possible reformulations: convex hull (chull), big M method (bigM) or indicator constraints (indic). Note that in the the sequencing model [SEQUENCE] all three options are implemented. Note further, that currently, indicator constraints can only be handled by the solvers CPLEX, SCIP and XPRESS.

Note
- By default, the convex hull reformulation method is used.
- A different reformulation method may be used for each disjunction.
Observe that for the convex hull reformulation, the value of the parameter \( \text{epsilon} \) may optionally be specified. This parameter is an upper bound to check for constraint satisfaction, the default value is 0.0001.

Note that for the big M method, the value of \( M \) may optionally be specified. The value of \( M \) should be large enough to relax the constraint, but it should not be too large, to avoid infeasible solutions. The default value is 10000.

Following the EMP keyword disjunction, the first mandatory entry is a specification of the variable that governs the disjunction: \([\text{NOT}] \text{ var}\)\(\ast\). This may be either a binary variable, a negated binary variable or the symbol \(\ast\), that is internally replaced by default binary variables. For an example and more details on the symbol \(\ast\) in this context, see section Disjunctive Programming with EMP: Example with No Logic Propositions. Further, \{equ\} denotes a set of GAMS equation names that must be satisfied if the first disjunction term is selected. The remainder of the syntax is self-explanatory.

Alternatively, the following syntax may be used:

Default \[\text{chull [chull eps]} \mid \text{bigM [big M value]} \mid \text{indic}\]

Disjunction \[\text{NOT} \text{ var}\ast \text{ NOT} \{\text{equ}\} \text{ ELSEIF NOT} \text{ var}\ast \\text{ NOT} \{\text{equ}\} \text{ ELSE NOT} \{\text{equ}\}\]

Note that the first line is optional and serves to specify the reformulation method. The second line is identical to the first formulation of the general syntax where the specification of the reformulation method is omitted. Some users find this alternative syntax clearer, since the specification of the reformulation method and the disjunction are separated.

In addition to the sequencing model \([\text{SEQUENCE}]\) that was already mentioned, the GAMS EMP Library has two other models for disjunctive programming: the manufacturing problem \([\text{FOODEMP}]\) and the job scheduling problem \([\text{MAKESPAN}]\).

### 4.51.11 Stochastic Programming

Stochastic programs are mathematical programs that involve data that is not known with certainty. Deterministic programs are formulated with fixed parameters, whereas real world problems frequently include some uncertain parameters. Often these uncertain parameters follow a probability distribution that is known or can be estimated. Thus stochastic programs approximate unknown data by probability distributions. The goal is to find some policy that is feasible for all (or almost all) the possible data instances and that maximizes the expectation of some function of the decision variables and the random variables. The EMP framework includes an extension for stochastic programming that allows users to model various stochastic problems as deterministic models, while information about the stochastic structure of the problem, like probability distributions for some data, is specified in the EMP annotations. Thus formulating stochastic programs becomes straightforward.

In the remainder of this chapter we discuss the stochastic programming extension of GAMS EMP. We introduce the basics of stochastic programming with EMP using a two-stage stochastic model and then show how the logic can be extended to multi-stage stochastic problems. In most stochastic problems the expected value of the objective is optimized. The EMP framework facilitates optimizing two additional risk measures: Value at Risk (VaR) and Conditional Value at Risk (CVaR). These alternative risk measures are discussed next. Another type of stochastic programming includes constraints that hold only with certain probabilities. These constraints are called chance constraints (or probabilistic constraints), they are the topic of the last section on stochastic programming. At the end of each section we give an overview of the EMP annotations that are specific to each topic: stochastic programming with recourse, additional risk measures, and chance constraints. A summary of all EMP keywords for stochastic programming is given in section GAMS EMP Keywords for Stochastic Programming.
4.51.11.1 Stochastic Programming with Recourse

One way to think about stochastic problems is to require the decision maker to make a decision now and then to minimize the expected costs of the consequences of that decision. This paradigm is called the recourse model. The simplest form of the recourse model has two stages: a decision is made in the first stage, then the realization of the uncertain parameters is revealed at the start of the second stage and recourse actions can be taken given this new information. This simple model can be extended to include more stages. In a multistage problem, a decision is made in the first stage, then some uncertainty is resolved in the second stage and another decision is made based on this new knowledge, then some other uncertainty is resolved and so on. The objective is to minimize the expected costs of the decisions in all stages.

This section is organized as follows. We start with a mathematical formulation of the two-stage stochastic problem with recourse, then show how such problems can be modeled with EMP using a simple example. The uncertain data in this first example follows a discrete distribution, there are just three different scenarios. Continuous distributions are more complex to model. Mostly, they are approximated to discrete distributions by sampling, which is discussed next. The second example illustrates how the modeling approach for a simple two-stage problem can be extended to solve a multi-stage problem. We conclude this section with a description of the EMP annotations for stochastic programs with recourse.

4.51.11.1.1 Two-Stage Stochastic Programs: Mathematical Formulation

The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let \( x \in \mathbb{R}^n \) and \( y \in \mathbb{R}^m \) be two variables and let the set of all realizations of the unknown data be given by \( \Omega \), \( \Omega = \{\omega_1, \ldots, \omega_S\} \subseteq \mathbb{R}^r \), where \( r \) is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

\[
\begin{align*}
\text{Min}_x & \quad z = c^T x + \mathbb{E}[Q(x, \omega)] \\
\text{s.t.} & \quad A x = b, \quad x \geq 0,
\end{align*}
\]

where \( Q(x, \omega) = \text{Min}_y \) (31)

\[
\begin{align*}
\text{s.t.} & \quad T \omega x + q^T \omega y(\omega) = h, \quad y(\omega) \geq 0, \quad \forall \omega \in \Omega.
\end{align*}
\]

The first two lines define the first-stage problem and the last two lines define the second-stage problem. In the first stage, \( x \) is the decision variable, \( c^T \) represents the cost coefficients of the objective function and \( \mathbb{E}[Q(x, \omega)] \) denotes the expected value of the optimal solution of the second stage problem. In addition, \( A \) denotes the coefficients and \( b \) the right-hand side of the first stage constraints. In the second stage, \( y \) is the decision variable, \( T \) represents the transition matrix, \( W \) the recourse matrix (cost of recourse) and \( h \) the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on the specific realization of the stochastic data \( \omega \). The objective variable \( z \) is also a random variable, since it is a function of \( \omega \). As a random variable cannot be optimized, stochastic solvers automatically optimize the expected value of the objective variable \( z \). Note that the EMP framework allows other risk measures to be optimized in addition to the expected value. This is discussed below.

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then it will become known which values the uncertain parameters will take. Given the fixed solution of the first stage and the new data, recourse action can be taken in the second stage and the optimal solution determined. Each possible realization of the uncertain data is represented by \( \omega_s \in \Omega \) and is called a scenario. The objective is to find a feasible solution \( x \) that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the deterministic equivalent. Assume that the uncertain parameters follow a (finite) discrete distribution and that each scenario \( \omega_s \) occurs with probability \( P(\omega_s) = p_s \) for all \( s = 1, \ldots, S \) and \( \sum_s p_s = 1 \). Thus \( \mathbb{E}[Q(x, \omega)] = \sum_s p_s Q(x, \omega_s) \).
\[
\sum s_i p_s q^T y_s, \quad \text{where } y_s \text{ denotes the optimal second-stage decision for the scenario } \omega_s. \text{ Then the deterministic equivalent can be expressed as follows:}
\]

\[
c^T x + p_1 q^T y_1 + p_2 q^T y_2 + \ldots + p_S q^T y_S \\
\text{s.t.} \\
Ax + W_1 y_1 \quad = \quad b \\
T_1 x + W_2 y_2 \quad = \quad h_1 \\
\vdots \quad \vdots \\
T_S x + W_S y_S \quad = \quad h_S \\
x \in \mathbb{R}^n \quad y_1 \in \mathbb{R}^m \quad y_2 \in \mathbb{R}^m \quad + \quad W_S y_S \quad = \quad h_S
\]

Note that for stochastic linear programs the deterministic equivalent is just another (potentially very large) linear program.

A GAMS EMP stochastic model has three parts: the core model, EMP annotations and the dictionary, which contains output-handling information. The core model is a deterministic model, where the uncertain data is given as fixed parameters. Often, the expected value of the probability distribution is chosen.

The annotations contain information about the stochastic features of the model: a specification of the random variables and their distributions, details about the stages and possibly directions concerning sampling.

Given the probability distributions of the random variables, the solvers of stochastic programming models create various scenarios and evaluate them. In the dictionary users specify which results from each scenario are to be stored in standard parameters in order to be accessed later. These three parts of a GAMS EMP stochastic model will become much clearer as they are illustrated in the next subsection.

### 4.51.11.1.2 A Simple Example: The News Vendor Problem

Consider the following simple example adapted from the news vendor model [NBSIMPLE]. A news vendor has to decide early in the morning how many newspapers to buy from a distributor on a particular day in order to sell them to his customers. He knows from experience that the demand will be 45 in 70% of all cases, 40 with a probability of 20% and 50 with a probability of 10%. If the demand is less than the number of newspapers he bought in the morning, the left-over newspapers will be stored in an inventory at a cost per unit. If the demand exceeds his expectations and there are more customers than newspapers, then he will have to pay a penalty. He aims to maximize his profit by selling the newspapers at a higher price than he has bought them. Mathematically, the problem can be expressed as follows:

\[
\text{Max}_x \quad Z(x, D) = -cx + E[Q(x, D)], \quad x \geq 0,
\]

where

\[
Q(x, D) = \max_{s,i,l} \quad vs_D - hi_D - pl_D \\
\text{s.t.} \quad x - s_D - i_D = 0 \\
\quad s_D + l_D = D, \quad s_D, i_D, l_D \geq 0
\]

where

- the variable \(x\) denotes the number of newspaper bought in the morning,
- \(c\) is the cost per newspaper,
- \(D\) is a random variable that denotes the uncertain demand; the set of all realizations of \(D\) is given by \(\Omega = \{d_1, d_2, d_3\}\), with \(d_1 = 45\), \(d_2 = 40\) and \(d_3 = 50\), and the respective probabilities \(p_1 = 0.7\), \(p_2 = 0.2\) and \(p_3 = 0.1\),
- the variable \(s\) denotes the number of newspapers that are sold,
\( v \) is the selling price,

- the variable \( i \) denotes the number of newspapers that could not be sold, in case the demand turned out to be less than expected,

- \( h \) is the holding cost in the inventory per unit,

- the variable \( l \) denotes the unsatisfied demand, in case the demand turned out to be higher than expected,

- \( p \) is the penalty per unit of unsatisfied demand.

Note that \( x \) is the first-stage decision variable and the variables \( s \), \( i \) and \( l \) are the second-stage decision variables. The second-stage decision variables are dependent on \( D \). An overview of the stages in this problem is given in the following table:

<table>
<thead>
<tr>
<th>1st Stage Decision Variable</th>
<th>2nd Stage Decision Variable; ( \Omega = { d_1, d_2, d_3 } )</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y_{d_1} = (s_{d_1}, l_{d_1}, i_{d_1}) )</td>
<td>Scenario 1: ( p_1 = 0.7 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( y_{d_2} = (s_{d_2}, l_{d_2}, i_{d_2}) )</td>
<td>Scenario 2: ( p_2 = 0.2 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3}) )</td>
<td>Scenario 3: ( p_3 = 0.1 )</td>
</tr>
</tbody>
</table>

Table 3: Two stages in the News Vendor Problem

Observe that the decision \( x \) has to be made before the realization of the demand \( D \) is known. Given the uncertainty of the demand, we aim to maximize the expected value of the profit, denoted by \( \mathbb{E}[Z(x, D)] \). The expected value of the profit is the profit on average. Note that as we have a finite number of scenarios and their probabilities are known, the expected value of the profit \( \mathbb{E}[Z(x, D)] \) can be expressed as a weighted sum:

\[
\mathbb{E}[Z(x, D)] = -cx + \mathbb{E}[Q(x, D)] - cx + \sum_{k=1}^{3} p_k Q(x, d_k).
\] (34)

Note that in this example there are no first stage equations \( Ax = b \).

The core model is the first part of the respective GAMS EMP model:

```
Variable z "profit";
Positive Variables
  x "units bought"
  i "inventory"
  l "lost sales"
  s "units sold";
Scalars c "purchase costs per unit" / 30 /
p "penalty shortage cost per unit" / 5 /
h "holding cost per leftover unit" / 10 /
v "revenue per unit sold" / 60 /
d "demand, random parameter" / 45 /;
Equations profit "profit to be maximized"
row1 "demand = UnitsSold + LostSales"
row2 "inventory = UnitsBought - UnitsSold";
```
profit..  z =e= v*s - c*x - h*i - p*l;
row1..  d =e= s + l;
row2..  i =e= x - s;

Model nv / all /;

Observe that the model is defined in the usual way. In particular, the demand \(d\) is modeled as a fixed parameter.

The EMP annotations are the second part. Here we specify the stochastic structure:

File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar d discrete 0.7 45 0.2 40 0.1 50
stage 2 i l s d
stage 2 Row1 Row2
$offput
putclose emp;

First, we define the parameter \(d\) to be a random variable using the EMP keyword randvar and we specify the probability distribution. The EMP keyword discrete indicates that \(d\) follows a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50.

Note

If the sum of the probabilities of a discrete distribution is smaller or larger than 1, the EMP framework will automatically normalize the probabilities so that the sum equals 1 and a corresponding remark will appear in the log and listing file.

Observe that continuous distributions are also possible, see section Random Variables with Continuous Distributions below for details.

Secondly, we specify which variables and equations belong to stage 2 using the EMP keyword stage. Note that the variables and equations that are not listed in the annotations are automatically assigned to a stage, by default stage 1 (this could be changed using the EMP keyword stageDefault). Only the objective variable (in this case \(Z\)) and the objective equation (profit) are assigned to the highest stage specified instead (stage 2 in this example). Observe that \(Z\) is in fact a random variable since it is a function of the random variable \(D\). As such it cannot be optimized directly, EMP implicitly maximizes the expected value of \(Z\). This might lead to some confusion since the expected value of \(Z\) belongs to stage 1. We show in section Expected Value Revisited how to specify more clearly the fact that we are maximizing \(E(Z)\).

All keywords that can be used in EMP annotations in the context of stochastic programming are introduced in subsequent examples and they are summarized in section GAMS EMP Keywords for Stochastic Programming.

The dictionary with output-handling information and the solve statement are the third part of the model. After solving a stochastic programming model, only the solution of the expected value problem may be accessed via the regular .l and .m fields. The results of the scenarios that were created and evaluated by the stochastic solvers may be stored in standard parameters in the following way:
Set scen "scenarios" / s1*s3 /;
Parameter
  s_d(scen) "demand realization by scenario"
  s_x(scen) "units bought by scenario"
  s_s(scen) "units sold by scenario"
  s_rep(scen,*) "scenario probability" / #scen.prob 0/;

Set dict / scen .scenario.''
  d .randvar .s_d
  s .level .s_s
  x .level .s_x
  '' .opt .s_rep /;

solve nv max z use EMP scenario dict;
display s_d, s_x, s_s, s_rep;

The size of the set scen defines the maximal number of scenarios we are willing to store results for. It does not have to match the number of scenarios that are actually generated in the solution process. Assume that the size of the set scen is \( n \) and \( n \) is smaller than the number of generated scenarios. In this case only the results for the first \( n \) scenarios will be stored. On the other hand, if the size of scen is larger than the number of generated scenarios, then the positions of the surplus elements of scen will be empty in the parameters (e.g. s_d).

The elements of the three-dimensional set dict determine which scenario-dependent values will be stored. The set contains mapping information between the symbols in the model in the first position and the symbols in which scenario solution information is stored in the third position. The type of storing is specified in the second position. The following entries are allowed:

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Stores the level values of a scenario solution of a variable or equation.</td>
</tr>
<tr>
<td>marginal</td>
<td>Stores the marginal values of a scenario solution of a variable or equation.</td>
</tr>
<tr>
<td>randvar</td>
<td>Stores the realization of a random variable.</td>
</tr>
<tr>
<td>opt</td>
<td>Stores the probability of each scenario.</td>
</tr>
<tr>
<td>scenario</td>
<td>The symbol in the first position of the tuple is used as the scenario index.</td>
</tr>
</tbody>
</table>

**Table 4** Entries in the second position of the tuples in the scenario dictionary

**Attention**

The first two tuples in the set dict are mandatory, all other elements of the set are optional. If one of the first two tuples is missing, the stochastic model cannot be solved and an error will be generated.

In the example above, we store the realization for each scenario of the random variable d in the parameter s_d, the level values of the variables s and x in the parameters s_s and s_x respectively and the actual probability of each scenario in the parameter s_rep.

**Note**

Storing and retrieving the actual probabilities of the scenarios may be particularly relevant if the probabilities have been normalized such that their sum equals 1.

Finally, the solve statement needs to be adjusted: we use the model type EMP and add scenario dict to indicate that a stochastic problem is to be solved.
Random Variables with Continuous Distributions

Now we assume that the random variable $D$ in the example above has a continuous distribution, say a Normal distribution with mean 45 and standard deviation 10. The structure of the problem remains unchanged, the only difference is that the set $\Omega$, that is the set of all realizations of $D$, contains an infinite number of scenarios. There are various ways of modeling this. For example, a sampling procedure implemented in the solver can be used: a finite number of scenarios is generated to approximate $\Omega$ and thus the problem is converted to a problem with a finite discrete distribution.

We can model a continuous distribution of the random variable in GAMS EMP by changing the annotations in the following way:

```
randvar d normal 45 10
```

Note that currently only the solver LINDO has implemented a sampling procedure for parametric distributions so that it can solve such a problem directly. If the solver DE is used, the sampling can be performed by the LINDO system behind the scenes. More details about sampling are given in the next section. All parametric distributions that can be modeled are listed in Table 5.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>shape 1</td>
<td>shape 2</td>
<td></td>
</tr>
<tr>
<td>Cauchy</td>
<td>location</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Chi_Square</td>
<td>deg. of freedom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>lambda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>deg. of freedom 1</td>
<td>deg. of freedom 2</td>
<td></td>
</tr>
<tr>
<td>Gamma</td>
<td>shape</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Gumbel</td>
<td>location</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Laplace</td>
<td>mean</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Logistic</td>
<td>location</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>LogNormal</td>
<td>mean</td>
<td>std dev</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>mean</td>
<td>std dev</td>
<td></td>
</tr>
<tr>
<td>Pareto</td>
<td>scale</td>
<td></td>
<td>shape</td>
</tr>
<tr>
<td>StudentT</td>
<td>deg. of freedom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangular</td>
<td>low</td>
<td>mid</td>
<td>high</td>
</tr>
<tr>
<td>Uniform</td>
<td>low</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>Weibull</td>
<td>shape</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Binomial</td>
<td>n</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>Geometric</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyper_Geometric</td>
<td>total</td>
<td>good</td>
<td>trials</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>p-factor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative_Binomial</td>
<td>failures</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>Poisson</td>
<td>lambda</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5**: Parametric distributions supported by LINDO

Recall that there are two ways to specify that a specific solver should be used (that is not the default solver for the problem type): with a command line parameter (e.g. `emp=lindo`) and with an option statement (e.g. `option emp = lindo;`).

Examples with random variables that follow parametric distributions include the multistage example below (three random variables with Gamma distributions) and the production model \[\text{PRODSP3}\] in the GAMS EMP Model Library (one random variable that follows a Normal distribution and one random variable that is uniformly distributed).
4.51.11.1.4 Sampling  Currently, only the solver LINDO has the ability to perform sampling for parametric distributions directly. Note that LINDO generates 6 samples by default and users need a LINDO license for sample sizes larger than 10 (smaller sample sizes are included in the demo version, but just for certain distributions). Also, changing the default variance reduction method requires a valid LINDO license. There are three ways to customize sampling: including additional information in the EMP annotations, generating a sample with the LINDO library lsadclib (these two options allow to pass on the samples to other solvers with EMP SP capabilities) and setting various options in the solver LINDO. We discuss each method in more detail in the next three subsections.

4.51.11.1.4.1 Customizing Sampling in the EMP Annotations  The EMP framework provides two keywords to facilitate sampling: sample and setSeed. The keyword sample allows users to specify the size of the sample in the EMP annotations. Consider the following example:

```plaintext
randvar d normal 45 10
sample d 9
```

The second line specifies that the size of the sample of the distribution of the random variable $D$ is 9. Note that currently the LINDO Sampling Library is used for this sampling. If the solver DE is used and a parametric distribution for the random variable is specified, the second line with the keyword sample is mandatory. Otherwise the following error message will appear:

```plaintext
*** Only random variables with sampled continuous distributions supported.
```

In addition to specifying the size of the sample, information about a mathematical variance reduction method may be added in the line that starts with the keyword sample. Variance reduction is a procedure used to increase the precision of the estimated values from the distribution. LINDO provides three methods for reducing the variance: Monte Carlo sampling, Latin Square sampling and Antithetic sampling. We illustrate with an example.

Consider a stochastic model with four random variables: $E$, $F$, $G$ and $H$. Assume that $E$ follows a Normal distribution with mean 23 and standard deviation 5, $F$ follows a Normal distribution with mean 37 and standard deviation 8, $G$ is uniformly distributed on the interval $[0, 1]$ and $H$ is binomially distributed with $n = 100$ and $p = 0.55$. We wish that three variance reduction methods are applied: Antithetic sampling is to be used for $E$ and $F$, Monte Carlo sampling for $G$ and Latin Square sampling for $H$. We insert the following lines in the EMP annotations:

```plaintext
randvar e normal 23 5
randvar f normal 37 8
randvar g uniform 0 1
randvar h binomial 100 0.55
sample e f 10 method1
sample g 12 method2
sample h 8 method3
```

First, the random variables and their distributions are defined. Next, details about the sampling procedures are given. Note that the keyword sample can take more than one random variable if the sample size and the variance reduction method for these random variables are identical. We need to add the following lines before the solve statement to specify the content of method1, method2 and method3 (we assume that the name of the model is nv):

```plaintext
```
If Latin Square sampling should also be used for $E$ and $F$, we would simply change the EMP annotations to replace the label `method1` with the label `method3`. For more details on variance reduction methods, please consult the LINDO manual.

In addition, the EMP keyword `setSeed` may be used to further customize the sampling procedure:

```
setSeed <seed>
```

This line sets the seed (an integer number) for the random number generator of the sampling routines. If `setSeed` is used in the EMP annotations, the seed will be set once before all samples are generated. Please note that `setSeed` only works with a valid LINDO license.

### 4.51.11.4.2 Separating Sampling and Solving

Users may want to sample from a distribution with the LINDO system and solve the model with another solver, say DECIS. This is possible with the sampling routines from the LINDO Sampling Library `lsadclib`. We could solve the news vendor model by first drawing a sample from a Normal distribution with mean 45 and standard deviation 10 and then using the sample in the EMP annotations. The GAMS code for sampling from a Normal distribution where the sample size is 9 follows:

```
$funcLibIn msllib lsadclib

Function setSeed / msllib.setSeed /
   sampleNormal / msllib.sampleLSNormal /
   getSampleValues / msllib.getSampleValues /;

Scalar k;
   k = sampleNormal(45,10,9);

Set g /1*9/; parameter sv1(g);
   loop(g,
      sv1(g) = getSampleValues(k);
   );
   display sv1;
```

The directive in the first line makes the LINDO sampling library available, `msllib` is the internal library name (see also `$funcLibIn` for more information). For further details and a list of the available probability distributions, see section The LINDO Sampling Library.

In the following lines we demonstrate how the sample is used in the EMP annotations:

```
File emp / '%emp.info%' /;
put emp;
   put 'randvar d discrete '; loop(g, put (1/card(g)) ' ' sv1(g) ' ');
$onput
   stage 2 I L S d
   stage 2 Row1 Row2
$offput
putclose emp;
```

Note that the third line states that the random variable $D$ follows a discrete distribution and the probabilities and values are taken from the previously generated sample. The other lines of the annotations remain unchanged.
4.51.11.1.4.3 Sampling Options for the Solver LINDO  There are some customizable sampling options in LINDO. Users could control the number of sampled scenarios by setting any of the following LINDO/SP options in the file LINDO options file lindo.opt:

- `STOC_NSAMPLE_PER_STAGE` - list of sample sizes per stage (starting at stage 2)
- `STOC_NSAMPLE_SPAR` - common sample size per stochastic parameter
- `STOC_NSAMPLE_STAGE` - common sample size per stage

For example, we could insert the following three lines before the solve statement:

```
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nv.optfile = 1;
```

The first line directs GAMS to solve models of the type `emp` with the LINDO solver. The second line writes `STOC_NSAMPLE_STAGE = 100` to the file `lindo.opt`, it communicates to the solver to generate 100 samples per stage. The third line informs GAMS to use the solver option file (i.e. `lindo.opt`).

The following example shows how to use large samples and instruct LINDO to use a Benders decomposition algorithm to tackle the problem:

```
$onecho > lindo.opt
STOC_MAX_NUMSCENS = 1000000
STOC_NSAMPLE_STAGE = 40
STOC_METHOD = 1
$offecho
```

In the second line we ensure that the maximum number of scenarios is large enough. The options in the following lines state the sample size and determine the stochastic method to be used (1 means Nested Benders Decomposition). For further details on LINDO options, please consult the LINDO user manual.

4.51.11.1.5 Multistage Stochastic Programming Example  The modeling principles for two-stage stochastic models can be easily extended to multistage stochastic models. At the beginning of each stage some uncertainty is resolved and recourse decisions or adjustments are made after this information has become available. At the point where decisions are made only outcomes of the current stage and previous stages are available. This logic can be pictured schematically as follows:

```
Make decision \rightarrow \text{Random Variable is realized} \rightarrow \text{Make decision} \\
\text{Stage 1} \rightarrow \cdots \rightarrow \text{Stage 2} \rightarrow \cdots \rightarrow \text{Stage n}
```

Observe that random variables which are realized in stage $k$ are fixed parameters in stage $k + 1$ and following; stage 1 random variables are in fact simply given deterministic values and will create a warning or an error dependent on the solver selected.

Consider an inventory problem. At the end of each week the decision how many hats should be bought in order to satisfy the stochastic demand in the following week must be made. The aim is to maximize the profit. We assume that the stochastic demand can be modeled using a Gamma distribution. The planning horizon is 3 weeks. Before the first week starts an initial purchase decision has to be made and
the goods are stored in the inventory for use in week 1. At this point only the distribution of the demand of the first week is known. During the first week the actual demand is revealed and some items that were stored in the inventory are sold. Some items may be left over; they are stored as the inventory for the second week. In addition, a purchase decision for the second week has to be made given the size of the inventory and the distribution of the demand in the second week. Again, the actual demand is revealed in the course of the second week. The same holds for the third week.

We will model the problem with 4 stages, where the first stage corresponds to the preparation time before the first week, the second stage corresponds to decisions made in the first week, the third stage corresponds to decisions made in the second week and the fourth stage corresponds to decisions made in the third week. Let \( t \) denote the stages. Note that while the stages range from \( t = 1 \) to \( t = 4 \), demand variables are realized only at \( t = 2 \) to \( t = 4 \). Let \( y_t \) be the amount bought and \( i_t \) the amount stored at the end of each stage and let \( s_t \) denote the demand in each week and \( s_t \) the amount sold each week. Note that we denote the demand with a capital \( D_t \) since it is a random variable. Let \( \alpha = 10 \) be the cost per hat bought, \( \beta = 20 \) the revenue per hat sold and \( \delta = 4 \) the storage cost per unit. Further, in the storage facility a maximum of \( \kappa = 5000 \) hats can be stored.

Mathematically, this problem can be expressed as follows:

\[
\begin{align*}
\text{Maximize} & \quad z = -\alpha y_1 - \gamma i_1 + E[ \text{Max} \ eta s_2 (D_2) - \alpha y_2 (D_2) - \delta i_2 (D_2) + \cdots + \\
& \quad E[ \text{Max} \ eta s_4 (D_4) + -\alpha y_4 (D_4) - \delta i_4 (D_4) ] ] \\
\text{subject to} & \quad i_1 = y_1 \\
& \quad i_t - 1 + y_t = s_t + i_t \\
& \quad s_t \leq i_{t-1} \\
& \quad s_t \leq D_t \\
& \quad i_t \leq \kappa \\
& \quad y_1, i_1, s_1, y_t, i_t \geq 0,
\end{align*}
\]

where \( t = 2, \ldots, 4 \) and \( D_t \) follows a Gamma distribution. Note that \( s_t, y_t \) and \( i_t \) depend on the realization of \( D_t \).

As discussed above, the solvers use a sampling procedure to approximate a problem with a continuous random variable by a problem with a discrete distribution. Figure 1 illustrates the stages assuming a sample size of 6 (per stage). Note that this results in a total of \( 6^3 = 216 \) scenarios.

The problem can be modeled with GAMS EMP as follows:

```plaintext
* Core Model
Set t "stages" /1*4/;
Set st(t) "stages where sales occur" /2*4/;
Positive Variables
  y(t) "units to be bought in time period t"
  i(t) "ending inventory in period t"
  s(t) "units sold in time period t"
Free Variable profit ;
Scalars
  kappa "capacity of storage building" /5000/
  alpha "cost per unit bought" /10/
  beta "revenue per unit sold" /20/
  delta "cost per unit stored at the end of time period" /4/;
Parameters
  k "shape of demand (1st parameter of gamma distribution)" /16/
  d_theta(st) "scale of demand (2nd parameter of gamma distribution)" /2 208.3125/
```

Figure 4.2 Stages in the inventory model

\begin{align*}
\text{d}(\text{st}) & \quad \text{"demand";} \\
\text{d}(\text{st}) & = k \times \text{d}_{\text{theta}}(\text{st});
\end{align*}

Equations

\begin{align*}
\text{defprofit} & \quad \text{"definition of profit"} \\
\text{balance}(t) & \quad \text{"balance equation"} \\
\text{sales1(st)} & \quad \text{"sales cannot exceed demand"} \\
\text{sales2(t)} & \quad \text{"sales cannot exceed inventory of previous time period"} \\
\end{align*}

\begin{align*}
\text{defprofit} & \quad \text{profit} = \sum(t, \beta \times \text{s}(t)\text{st}(t) - \alpha \times \text{y}(t) - \delta \times \text{i}(t)) \\
\text{balance}(t) & \quad \text{i}(t-1) + \text{y}(t) = \text{s}(t)\text{st}(t) + \text{i}(t) \\
\text{sales1(st)} & \quad \text{s}(\text{st}) = \text{d}(\text{st}) \\
\text{sales2(t)} & \quad \text{s}(t) = \text{i}(t-1); \\
i\.up(t) & = \kappa;
\end{align*}

Model inventory /all/;

* EMP Annotations

File emp / '%emp.info%' /;
put emp; emp.nd=6;
put "randvar d('2') gamma ", k d_{\text{theta}}('2') /;
put "randvar d('3') gamma ", k d_{\text{theta}}('3') /;
put "randvar d('4') gamma ", k d_{\text{theta}}('4') /;
$onput
\begin{align*}
\text{stage 1 } & y('1') \text{i('1')} \text{balance('1')}
\text{stage 2 } & y('2') \text{d('2')} \text{s('2')} \text{i('2')} \text{balance('2')} \text{sales1('2')} \text{sales2('2')}
\text{stage 3 } & y('3') \text{d('3')} \text{s('3')} \text{i('3')} \text{balance('3')} \text{sales1('3')} \text{sales2('3')}
\text{stage 4 } & y('4') \text{d('4')} \text{s('4')} \text{i('4')} \text{balance('4')} \text{sales1('4')} \text{sales2('4')}
\end{align*}
$offput
* Dictionary

Set scen "scenarios" / s1*s216 /;
Parameters
  s_d(scen,st) "demand realization by scenario"
  s_y(scen,t) "units bought by scenario"
  s_s(scen,t) "units sold by scenario"
  s_i(scen,t) "units stored by scenario";

Set dict /
  scen .scenario.''
  d .randvar .s_d
  s .level .s_s
  y .level .s_y
  i .level .s_i /;

option emp=lindo;
solve inventory max profit using emp scenario dict;
display s_d, s_s, s_y, s_i;

Observe that in the core problem the values of the demand \( d(t) \) are replaced by the expected values of the random variable \( D_t \) which follows a Gamma distribution. Note that as expected, in stage 1 we have only the variables \( y \) and \( i \), but no variables \( s \) and \( d \). Recall that currently only the solver LINDO can solve models with parametric distributions, see sections Random Variables with Continuous Distributions and Sampling for more information.

4.51.11.1.6 EMP Syntax for Stochastic Programs with Recourse

The general syntax of the EMP annotations used to specify stochastic problems with recourse is as follows:

```
[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val val {val} {prob val val {val}}]
[setSeed number]
[sample rv1 [rv2 ... rvn] sampleSize [varRedMethod]]
[stage stageNo rv | equ | var {rv | equ | var}]
{stage stageNo rv | equ | var {rv | equ | var}]
[stageDefault stageNo]
```

The first three lines present three ways to specify random variables: a single random variable with a discrete distribution, a single random variable with a parametric distribution and joint random variables with discrete distributions. Note that `randvar`, `jrandvar` and `discrete` are EMP keywords. The distribution of single discrete random variables is defined by pairs of the probability `prob` of an outcome and the corresponding realization `val`, see example above. The distribution of parametric random variables is defined by the name of the distribution `distr` and the respective parameter(s) `par`. An overview of all supported parametric distributions can be found in Table 5. All possible values for `distr` and the related parameters `par` are listed there. The keyword `jrandvar` is used to define discrete random variables that are jointly distributed. At least two random variables must be named. For an example, see the news vendor model [NBDISCJOINT].
At least one random variable must be defined and all three ways to define random variables may appear in the EMP annotations. See the scheduling model [Airlift] for an example with both, a discrete random variable and a continuous random variable.

The keywords setSeed and sample are optional. The seed for the random number generator may be set with setSeed. The random number generator is used for the sampling routines that are called with the keyword sample. If setSeed is used, the seed will be set once before all samples will be generated. The keyword sample is followed by the name of the respective random variable, the sample size (a number) and - optionally - a variance reduction method. Note that the random variable must have been previously declared to follow a parametric distribution. Note further, that the sample size of more than one random variable may be customized simultaneously. Observe that the default sample size is 6. For details on available variance reduction methods, see section Customizing Sampling in the EMP Annotations.

Note
- A valid LINDO license is required to use the keywords setSeed and sample. The keyword sample may be used with a demo or community version of the LINDO license, but it is limited to the Normal and Binomial distributions with a maximum sample size of 10 and the variance reduction method cannot be changed.
- If a parametric distribution is used with any solver but LINDO, the keyword sample is mandatory, see above for details.

With the keyword stage random variables, variables and equations are assigned to their respective stages, where StageNo defines the stage number. Note that the default stage for all random variables, equations and variables is 1, except for the objective equation and variable. The default for those is the highest stage in the problem.

With the keyword stageDefault one can change the default stage to which symbols get assigned if they are not listed with a stage keyword explicitly.

4.51.11.2 Risk Measures with EMP

The literature on stochastic programming usually assumes that the expected value of the objective is optimized. The EMP tool follows this trend and implicitly optimizes the expected value. However, there are other risk measures that could be taken into consideration and that are frequently used in practice, particularly in finance. For example, a fund manager might be more interested in the expected value of the 10% worst cases of the projected wins than in the expected value of the overall distribution. The expected value of a partition of the distribution at one tail is called Conditional Value at Risk. The EMP framework provides special keywords to facilitate optimizing this risk measure. On a more abstract level, risk measures can be understood as mechanisms to evaluate the effects of uncertainty in the underlying system on the outcomes of interest. They can be used to modify the distribution of outcomes.

Using the example of an investor who wishes to balance expected rewards and the risk of loss when she decides how to allocate assets in a portfolio, we explore how stochastic optimization problems involving risk measures can be modeled with EMP. Note that this example is adapted from the stochastic portfolio model [Portfolio]. For simplicity of exposition, we only describe two-stage models here. In the examples that follow, the period \((0, T)\) is the period between investing in a portfolio of assets and return from this portfolio.

We first present the example problem and introduce a new EMP keyword to model the expected value explicitly. Then we introduce and discuss Value at Risk (VaR) and Conditional Value at Risk (CVaR). At the end of this section, we give a summary of the EMP annotations that are specific to risk measures.

Note
- The stochastic extension of EMP facilitates the optimization of a single risk measure or a combination of risk measures (for example, the weighted sum of Expected Value and CVaR). In addition, the modeler can choose to trade off risk measures.
4.51.11.2.1 Expected Value Revisited Suppose an investor has the opportunity to invest a certain amount in three assets. She is given the probability distribution in Table 6 that links each asset with a possible return at time $T$. The question arises how she should allocate her funds between the three assets at time 0 in order to maximize her expected return at time $T$.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Probability</th>
<th>ATT</th>
<th>GMC</th>
<th>USX</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1/12</td>
<td>1.300</td>
<td>1.225</td>
<td>1.149</td>
</tr>
<tr>
<td>s2</td>
<td>1/12</td>
<td>1.103</td>
<td>1.290</td>
<td>1.260</td>
</tr>
<tr>
<td>s3</td>
<td>1/12</td>
<td>1.216</td>
<td>1.216</td>
<td>1.419</td>
</tr>
<tr>
<td>s4</td>
<td>1/12</td>
<td>0.954</td>
<td>0.728</td>
<td>0.922</td>
</tr>
<tr>
<td>s5</td>
<td>1/12</td>
<td>0.929</td>
<td>1.144</td>
<td>1.169</td>
</tr>
<tr>
<td>s6</td>
<td>1/12</td>
<td>1.056</td>
<td>1.107</td>
<td>0.965</td>
</tr>
<tr>
<td>s7</td>
<td>1/12</td>
<td>1.038</td>
<td>1.321</td>
<td>1.133</td>
</tr>
<tr>
<td>s8</td>
<td>1/12</td>
<td>1.089</td>
<td>1.305</td>
<td>1.732</td>
</tr>
<tr>
<td>s9</td>
<td>1/12</td>
<td>1.090</td>
<td>1.195</td>
<td>1.021</td>
</tr>
<tr>
<td>s10</td>
<td>1/12</td>
<td>1.083</td>
<td>1.390</td>
<td>1.131</td>
</tr>
<tr>
<td>s11</td>
<td>1/12</td>
<td>1.035</td>
<td>0.928</td>
<td>1.006</td>
</tr>
<tr>
<td>s12</td>
<td>1/12</td>
<td>1.176</td>
<td>1.715</td>
<td>1.908</td>
</tr>
</tbody>
</table>

**Table 6: Return by scenario**

Mathematically, the problem can be expressed as follows:

$$\begin{align*}
\text{Max} \quad & E[R] \\
\text{s.t} \quad & R = \sum_j w_j v_j \\
& \sum_j w_j = 1 \\
& w_j \geq 0,
\end{align*}$$

(36)

where the variable $R$ denotes the return and is a function of the random variable $v$, $E[R]$ is the expected return, $w_j$ is the weight associated with each asset $j$, and $v_j$ is the return of each asset $j$. The weights can also be interpreted as proportions of the amount to be invested, their sum must equal 1. Note that $w_j$ is the decision variable in this problem. Note further, that $v_j$ is a random variable that depends on which scenario is realized.

We present two different ways to model this problem in GAMS EMP: in the first version the expected value of the return is modeled *implicitly* and in the second version it is modeled *explicitly*. Both models have two stages, in the first stage the weights are chosen without knowing which scenario will be realized, in the second stage the 12 scenarios are taken into account. We start with the part of the code where the data is given. It is named `data.gms` and it will be included in all models in this section.

```
Sets j "assets" / att, gmc, usx /
   s "scenarios" / s1*s12 /;
Table vs(s,j) "scenario returns from assets"
   att  gmc  usx
  s1   1.300 1.225 1.149
  s2   1.103 1.290 1.260
  s3   1.216 1.216 1.419
  s4   0.954 0.728 0.922
  s5   0.929 1.144 1.169
  s6   1.056 1.107 0.965
```
Parameters
\[ v(j) \quad \text{"return from assets"} \]
\[ p(s) \quad \text{"probability'' / #s [1/card(s)]} / \;
\]
\[ v(j) = \sum(s, vs(s,j))/\text{card}(s); \]

The first model is similar to the two-stage example model above, the fact that the expected return is being maximized is not stated explicitly but is only implied:

```
* Core Model
$include data.gms
Variables
  r "value of portfolio under each scenario"
  w(j) "portfolio selection"
Positive Variables
  w;
Equations
  defr "return of portfolio"
  budget "budget constraint'';
  defr.. r =e= sum(j, v(j)*w(j));
  budget.. sum(j, w(j)) =e= 1;
Model portfolio / all /
* EMP Annotations
File emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%''
  / 'stage 2 v r defr'
  / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx");
putclose emp;
* Dictionary
Parameters
  s_v(s,j) "return from assets by scenario"
  s_r(s) "return from portfolio by scenario'';
Set
  dict / s .scenario.'''
  v .randvar .s_v
  r .level .s_r /
solve portfolio using emp max r scenario dict;
display s_v, s_r;
```

As usual, the core model is defined first and the specifications relating to the stochastic structure of the model are written to the file `emp.info`. Note that the statement `emp.nd=4` ensures that 4 decimal places
are used for the values of the parameter vs. In the first line of the EMP annotations, the variables v and r and the equation defr are assigned to the second stage. Note that the other variables and equations (in this case, the variable w and the equation budget) are automatically assigned to the first stage. In the second line of the annotations, the EMP keyword jrandvar is used to declare that v('att'), v('gmc') and v('usx') are joint random variables, i.e. they are jointly distributed (thus we have 12 scenarios and not 12*12*12=1728 scenarios). The respective probabilities and values are specified with a loop statement. Observe that the syntax of the solve statement suggests that the return r is maximized. However, r is in fact a random variable, since it is a function of random variables, so actually the expected return is maximized.

In the second model we introduce a new variable for the expected return, evr. This new variable is linked to the EMP keyword ExpectedValue in the annotations, thus it is made explicit that the expected value of the return is optimized.

* Core Model
$include data.gms

Variables
   r "value of portfolio under each scenario"
   w(j) "portfolio selection"
   evr "expected value of r"
   obj "objective variable";
Positive Variables
   w;
Equations
   defr "return of portfolio"
   budget "budget constraint"
   defobj "objective equation";

   defr.. r =e= sum(j, v(j)*w(j));
   budget.. sum(j, w(j)) =e= 1;
   defobj.. obj =e= evr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%
   / 'ExpectedValue r evr'
   / 'stage 1 obj defobj evr'
   / 'stage 2 v defr r'
   / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
   put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
   s_v(s,j) "return from assets by scenario"
   s_r(s) "return from portfolio by scenario";
Set dict / s .scenario."
   v .randvar .s_v
   r .level .s_r /
solve portfolio using emp max obj scenario dict;
In the EMP annotations, the EMP keyword `ExpectedValue` is followed by the variable `r` and the variable `evr`. In this way the variable `evr` is declared to be the expected value of the (random) variable `r`. Note that the new variables `evr` and `obj` belong to stage 1 and the variable `evr` is maximized (via `obj`). The remainder of the annotations and the code referring to the output-handling information in the dictionary are the same like in the first model. Both models have the same solution. We prefer the second model since the syntax is more explicit and clearer.

Observe that the keyword `ExpectedValue` is particularly useful if users wish to model an objective that is the weighted sum of several risk measures, thus trading-off different risk measures. An example with the weighted sum of the expected value and Value at Risk as objective and an example with the weighted sum of the expected value and Conditional Value at Risk as objective are given below.

4.51.11.2.2 Value at Risk (VaR) The Value at Risk (VaR) is the value of the distribution at a given cut-off point; for example, the value of the standard Normal distribution at the point $x$ such that 95% of the distribution is to the left. In this subsection we introduce the notion of VaR in more detail by developing a mathematical formulation, demonstrating how VaR can be modeled with GAMS EMP using a simple example and showing how to optimize the weighted sum of the expected value and VaR.

4.51.11.2.2.1 VaR: Mathematical Formulation Suppose $G(x, \xi)$ is a real valued function of the decision vector $x$ and a random data vector $\xi$ and that it denotes the loss function of a portfolio of assets. We aim to restrict potential losses and so we choose a portfolio composition such that the loss only exceeds a certain threshold $\gamma$ ($\gamma \in \mathbb{R}$) with a probability smaller or equal to $\alpha$, $\alpha \in (0, 1)$, where $\alpha$ is small. This condition can be modeled as a chance constraint and has the following form:

$$P(G(x, \xi) > \gamma) \leq \alpha \quad (37)$$

It is easy to see that this can be written in the following way:

$$P(G(x, \xi) - \gamma \leq 0) \geq 1 - \alpha. \quad (38)$$

Consider the random variable $Z_x := G(x, \xi) - \gamma$. For a given value of $x$, let $F_Z(z) := P(Z \leq z)$ be the cumulative distribution function of $Z$. Now, the point $x$ satisfies the constraint (37) if and only if $F_Z(0) \geq 1 - \alpha$. This is equivalent to saying that $x$ satisfies the constraint (39) if and only if $F_Z^{-1}(1 - \alpha) \leq 0$.

The (left-side) quantile $F_Z^{-1}(\theta)$ is called Value at Risk. It is denoted by $VaR_\theta(Z)$, i.e.

$$VaR_\theta(Z) = \inf\{t : F_Z(t) \geq \theta\}. \quad (39)$$

Hence constraint (38) can be written in the following equivalent form:

$$VaR_{1-\alpha}(G(x, \xi)) \leq \gamma. \quad (40)$$

Value at Risk as introduced in equation (39) refers to a percentile on the left tail of a distribution. From now on it will be denoted by $VaR_\theta(Z)$. When considering the Value at Risk at the right tail of the distribution, $\theta$ typically equals 0.9 or 0.95, it is denoted by $VaR_\theta(Z)$.

Figure 2 illustrates $VaR_{0.05}(Z)$, where $Z$ is normally distributed with mean $\mu = 0.645$ and standard deviation $\sigma = 1$.

The EMP framework provides the EMP keywords `varlo` and `varup` as a convenient alternative to chance constraints to model Value at Risk.
4.51.11.2.2 VaR with EMP: A Simple Example  

We continue to illustrate with the stochastic portfolio model that we have already used above. Consider that the investor might be interested in a strategy that maximizes the threshold at a certain cutoff, say at 10% on the left tail of the return curve. Mathematically, the problem can be expressed as follows:

\[
\begin{align*}
\text{Max} & \quad \text{VaR}_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]

where $\text{VaR}_\theta$ is the Value at Risk at the lower $\theta$th percentile.

In the EMP annotations of the corresponding GAMS model we introduce a new variable for VaR ($\text{varr}$), a scalar to specify the percentile we are interested in (theta) and the EMP keyword $\text{varlo}$:

```plaintext
* Core Model
$include data.gms
Variables
r "value of portfolio under each scenario"
w(j) "portfolio selection"
varr "value at risk of r"
obj "objective variable";
Positive Variables
w;
```
Scalar theta "relative volume" / 0.1 /

Equations
defr "return of portfolio"
budget "budget constraint"
defobj "objective equation" ;
defr.. r =e= sum(j, v(j)*w(j));
budget.. sum(j, w(j)) =e= 1;
defobj.. obj =e= varr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%' 
  / 'varlo r varr ' theta 
  / 'stage 1 obj defobj varr' 
  / 'stage 2 r defr v' 
  / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
  s_v(s,j) "return from assets by scenario"
  s_r(s) "return from portfolio by scenario" ;

Set
  dict / s .scenario."
  v .randvar .s_v
  r .level .s_r /

solve portfolio using emp max obj scenario dict;

The first line in the EMP annotations begins with the EMP keyword varlo. This line specifies that the variable varr is the Value at Risk relating to the random variable r and the scalar theta is the percentile (in range 0 to 1) we consider. The other lines in the annotations are similar to the second model above. Note that the objective equals the Value at Risk at the left tail, denoted by $VaR$. Observe that the EMP keyword varlo specifies VaR at the left tail of the probability distribution. For the right tail of the distribution, the EMP keyword to be used is varup.

Note

- The keyword var is identical to varup which refers to the right tail of the distribution.
- Currently only the solver DE supports the keywords for VaR.

Note further, that it is only appropriate to maximize $VaR_\alpha$ and minimize $VaR_\alpha$.

The implementation of the keywords varlo and varup is based on a mixed integer program similar to that described in section Example with a Single Chance Constraint. Observe that these implementations are likely to be hard and/or time consuming to solve. There is an option that allows users to customize the big M value (default is 1000) in the same manner that we outline below:
Observe that the EMP framework offers an alternative, shorter way to model VaR. There is no additional variable for VaR, hence in the EMP annotations, the EMP keyword \texttt{varlo} is only followed by \texttt{theta} and the objective variable in the solve statement is \texttt{r}. As \texttt{varlo} is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

\begin{verbatim}
File emp / '%emp.info%' /;
put emp '* problem %gams.i%' / 'varlo' 'theta'
   / 'stage 2 r defr v'
   / "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;
...
solve portfolio using emp max r scenario dict;
\end{verbatim}

Note that this alternative notation is shorter, but it is also opaque, therefore we recommend the first way.

### 4.51.11.2.2.3 Combining VaR and Expected Value

In a variation of the example above, consider an investor who aims to take into account both, the expected return and the Value at Risk of the return at a certain threshold \( \theta \). She combines the two risk measures and uses a scalar (\( \lambda \)) as a weight. A mathematical formulation of the problem reads as follows:

\[
\begin{align*}
\text{Max} & \quad \lambda \mathbb{E}[R] + (1 - \lambda) \text{VaR}_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]

where \( \mathbb{E}[R] \) is the expected value of the return and \( \text{VaR}_\theta \) is the Value at Risk at the \( \theta \)th percentile.

This problem can be modeled in GAMS EMP as follows:

\begin{verbatim}
* Core Model
$include data.gms
Variables
    r         "value of portfolio under each scenario"
    w(j)      "portfolio selection"
    varr      "value at risk of r"
    evr       "expected value of r"
    obj       "objective variable" ;
Positive Variables
    w ;
Scalars
    theta     "relative volume" / 0.1 /
    lambda    "weight EV versus VaR" / 0.2 /
Equations
\end{verbatim}
defr "return of portfolio"
budget "budget constraint"
defobj "convex combination of both risk measures" ;

defr..  r =e= sum(j, v(j)*w(j));
budget.. sum(j, w(j)) =e= 1;
defobj.. obj =e= lambda*evr + (1-lambda)*varr;

Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%' /
   'ExpectedValue r evr'
   'varlo r varr ' theta
   'stage 2 r defr v'
   'stage 1 defobj obj'
   "jrandvar v('att') v('gmc') v('usx')"
loop(s,
   put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
   s_v(s,j) "return from assets by scenario /s1.att 1/"
   s_r(s) "return from portfolio by scenario" ;

Set dict / s .scenario.''
   v .randvar .s_v
   r .level .s_r /;

solve portfolio_ext using EMP max obj scenario dict;

Note that we introduced a variable for the expected value of the return like in the example above where we modeled the expected return explicitly. Note further, that we added the parameter lambda and we defined the variable obj to be the weighted sum of the expected value of the return evr and the lower VaR varr. Observe that in the EMP annotations both keywords ExpectedValue and varlo are used. The other parts of the model remain unchanged.

4.51.11.2.3 Conditional Value at Risk (CVaR) The Conditional Value at Risk (CVaR) is the expected value of the left or right tail of the distribution; for example, the expected value of the left 5% of the standard Normal distribution. Note that CVaR is also called expected shortfall. In this subsection we introduce and discuss CVaR in more detail by presenting a mathematical formulation, demonstrating how CVaR can be modeled with GAMS EMP using a simple example and showing how to optimize the weighted sum of the expected value and CVaR.

4.51.11.2.3.1 CVaR: Mathematical Formulation \( CVaR_\alpha \) is the expected average return (in a given time period) given that we are in the \( (\alpha \times 100\%) \) left tail of the return distribution, where \( \alpha \in (0, 1) \). In other words, \( CVaR_\alpha \) is a mean of the left tail. For example, if we are interested in the 5% worst cases, i.e. \( \alpha = 0.05 \), \( CVaR_\alpha \) is the conditional expectation of the return, given the return is no greater than VaR.

Let \( \xi \) be a random variable with probability density function \( p(\xi) \), let \( G(\xi) \) be a function of the random variable \( \xi \) denoting the return of a portfolio of assets and let \( \alpha \in (0, 1) \) be a probability. Then the conditional value at risk of \( G(\xi) \) is defined as

\[
CVaR_\alpha(G(\xi)) = \frac{1}{\alpha} \int_{-\infty}^{VaR_\alpha} G(\xi) \cdot p(\xi) d\xi, \tag{43}
\]
where \( VaR_\alpha \) is the value at risk.

### 4.51.11.2.3.2 CVaR with EMP: A Simple Example

In the stochastic portfolio example that we have used throughout this section, the investor might be interested to make sure that in the worst cases she loses as little as possible. Thus she could consider only the worst 10% of possible cases and allocate her funds such that the expected mean return in these cases is maximized. Mathematically, the problem can be expressed as follows:

\[
\begin{align*}
\text{Max} & \quad CVaR_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]

where \( CVaR_\theta \) is the CVaR at the confidence level \( \theta \) at the left tail of the distribution.

Like in the case of Value at Risk above, we introduce a new variable for CVaR (cvarr), a scalar to specify the percentile we are interested in (theta) and the EMP keyword cvarlo in the EMP annotations:

* Core Model

\$include data.gms

Variables

\begin{align*}
\text{r} & \quad \text{"value of portfolio under each scenario"} \\
\text{w(j)} & \quad \text{"portfolio selection"} \\
\text{cvarr} & \quad \text{"conditional value at risk of r"} \\
\text{obj} & \quad \text{"objective variable"} ;
\end{align*}

Positive Variables

\begin{align*}
\text{w} ;
\end{align*}

Scalar

\begin{align*}
\text{theta} & \quad \text{"relative volume"} / 0.1 / ;
\end{align*}

Equations

\begin{align*}
\text{defr} & \quad \text{"return of portfolio"} \\
\text{budget} & \quad \text{"budget constraint"} \\
\text{defobj} & \quad \text{"objective equation"} ;
\end{align*}

\begin{align*}
\text{defr..} & \quad \text{r} =e= \sum(j, v(j) \cdot w(j)); \\
\text{budget..} & \quad \sum(j, w(j)) =e= 1; \\
\text{defobj..} & \quad \text{obj} =e= \text{cvarr};
\end{align*}

Model portfolio / all /;

* EMP Annotations

\$emp / '%emp.info%' /
put emp '* problem %gams.i%' /
    'cvarlo r cvarr' theta
    / 'stage 1 obj defobj cvarr'
    / 'stage 2 r defr v'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary

Parameters
s_v(s,j) "return from assets by scenario"
s_r(s) "return from portfolio by scenario";

Set dict / s .scenario.
   v .randvar .s_v
   r .level .s_r /;

solve portfolio using emp max obj scenario dict;

In the EMP annotations, the first line starts with the EMP keyword cvarlo. This line specifies that the variable cvarl is the left-tail CVaR relating to the random variable r and the scalar theta is the fraction (in range 0 to 1) we consider. Like in the previous section, the objective variable, the objective equation and the variable in the objective equation belong to the first stage while the equation that handles the random data and all its variables belong to the second stage. Observe that the objective equals the Conditional Value at Risk.

Note that the keyword cvarlo specifies CVaR at the left tail of the probability distribution. For the right tail of the distribution the keyword to be used is cvarup.

Note
- The EMP keyword cvar is identical to the keyword cvarup.
- Currently only the solver DE supports the keywords for CVaR.

The conditional value at risk denoting the mean of the right tail of the distribution can be denoted by $CVaR$ and is defined as:

$$CVaR_{\alpha}(G(\xi)) = \frac{1}{1-\alpha} \int_{\gamma_{\alpha}}^{\infty} G(\xi) \cdot p(\xi) d\xi.$$ (45)

Observe that it is only appropriate to maximize $CVaR_{\alpha}$ and minimize $CVaR_{\alpha}$. Furthermore, $CVaR_{\alpha}$ is a concave function and so should only be constrained using e.g.

$$CVaR_{\alpha} \geq \gamma$$

and $CVaR_{\alpha}$ is convex, so should only appear in constraints like

$$CVaR_{\alpha} \leq \gamma.$$ 

The EMP framework offers an alternative way to model CVaR. In this alternative formulation, there is no additional variable for CVaR, hence in the EMP annotations, the EMP keyword cvarlo is only followed by theta and the objective variable in the solve statement is r. As cvarlo is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

File emp / '%emp.info%' /
put emp '* problem %gams.i%'
   / 'cvarlo' theta
   / 'stage 2 r defr v'
   / "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

solve portfolio using emp max r scenario dict;

Note that this alternative notation is shorter, but it is considerably less clear, therefore we recommend the explicit notation of the first model.
4.51.11.2.3.3 Combining CVaR and Expected Value  

In a final variation on the portfolio example, we consider an investor who aims to take into account both, the expected return and the Conditional Value at Risk of the return at a certain threshold \( \theta \). She combines the two risk measures and uses a scalar (\( \lambda \)) to weigh the two summands. A mathematical formulation of the problem reads as follows:

\[
\text{Max} \quad \lambda \mathbb{E}[R] + (1 - \lambda) \text{CVaR}_\theta[R] \\
\text{s.t} \quad R = \sum_j w_j v_j \\
\sum_j w_j = 1 \\
w_j \geq 0, \tag{46}
\]

where \( \mathbb{E}[R] \) is the expected value of the return and \( \text{CVaR}_\theta \) is the CVaR at the confidence level \( \theta \).

This problem can be modeled with GAMS EMP as follows:

* Core Model
$sinclude data.gms$
Variables
\[
\begin{align*}
\text{r} & \quad \text{"value of portfolio under each scenario"} \\
\text{w(j)} & \quad \text{"portfolio selection"} \\
\text{cvarr} & \quad \text{"conditional value at risk of r"} \\
\text{evr} & \quad \text{"expected value of r"} \\
\text{obj} & \quad \text{"objective variable"} \\
\end{align*}
\]
Positive Variables
\[
\text{w} \\
\]
Scalars
\[
\begin{align*}
\text{theta} & \quad \text{"relative volume"} / 0.1 / \\
\text{lambda} & \quad \text{"weight EV versus CVaR"} / 0.2 /; \\
\end{align*}
\]
Equations
\[
\begin{align*}
\text{defr} & \quad \text{"return of portfolio"} \\
\text{budget} & \quad \text{"budget constraint"} \\
\text{defobj} & \quad \text{"convex combination of both risk measures"}; \\
\end{align*}
\]
\[
\begin{align*}
\text{defr..} & \quad \text{r} = \text{e} = \sum(j, v(j) \cdot w(j)); \\
\text{budget..} & \quad \sum(j, w(j)) = \text{e} = 1; \\
\text{defobj..} & \quad \text{obj} = \text{e} = \lambda \text{evr} + (1 - \lambda) \text{cvarr}; \\
\end{align*}
\]
Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
   / 'ExpectedValue r evr'
   / 'cvvarlo r cvarr' theta
   / 'stage 2 r defr v'
   / 'stage 1 defobj obj'
   / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
   put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
\[
\begin{align*}
\text{s_v(s,j)} & \quad \text{"return from assets by scenario"} \\
\text{s_r(s)} & \quad \text{"return from portfolio by scenario"} \\
\end{align*}
\]
Note that apart from the EMP keyword varlo being replaced by cvarlo, the code is identical to the code above, where we maximized the weighted sum of the expected value and VaR. Observe that by decreasing the value of $\lambda$ from 1 to 0 we can explore the results of an increasingly risk averse behavior.

4.51.11.4 EMP Syntax for Risk Measures

The EMP annotations where risk measures are used have the same general syntax like stochastic models with recourse. In addition, the following keywords may be used:

- **ExpectedValue** $rv$ $var$
- **varlo** $[rv]$ $var$ scalar
g - **varup** $[rv]$ $var$ scalar
c - **cvarlo** $[rv]$ $var$ scalar
d - **cvarup** $[rv]$ $var$ scalar

The **EMP keyword Expected Value** is used to state that the variable $var$ is the expected value of the random variable $rv$. Note that the random variable $rv$ must be defined to be a random variable in the EMP annotations.

The EMP keywords varlo, varup, cvarlo and cvarup refer to VaR at the left tail of the distribution ($VaR_{\alpha}$), VaR at the right tail of the distribution ($VaR_{1-\alpha}$), CVaR at the left tail of the distribution ($CVaR_{\alpha}$) and CVaR at the right tail of the distribution ($CVaR_{1-\alpha}$) respectively.

Note

The EMP keyword var is a synonym to varup and cvar is a synonym to cvarup.

The specifications that follow all four keywords have a long and a short version. In the long version, the random variable $rv$ and the variable that is assigned to be the respective risk measure (VaR or CVaR) are named explicitly. The scalar is a number in the interval $(0, 1)$, it defines the percentile in VaR and confidence level in CVaR. In the short version, only the scalar is specified. In this case the risk measure will applied to the objective variable implicitly.

Note

These five keywords are only supported by the solver DE.

4.51.11.3 Chance Constraints with EMP

In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints (or some of them) to be violated with a certain probability. Note that chance constraints are also called probabilistic constraints.

This section is organized as follows. After introducing a mathematical formulation of chance constraints, we show how such problems can be modeled with GAMS EMP using a simple example with only one chance constraint. Then we extend the model to include multiple chance constraints. We discuss the two ways to model problems with multiple chance constraints: using joint chance constraints and using individual chance constraints. Joint chance constraints require that all constraints are satisfied simultaneously with a given probability. Individual chance constraints require each constraint to be satisfied with a given probability independently of other constraints. Further, the EMP framework offers the option to penalize the violation of chance constraints. This is the topic of subsection Penalizing Violations of Chance Constraints. We conclude this section with an overview of the EMP annotations that are specific to chance constraints.
4.51.11.3.1 Chance Constraints: Mathematical Formulation  

Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

\[
\begin{align*}
\text{Min} & \quad c^T x \\
\text{s.t.} & \quad P(Ax \leq b) \geq p \\
& \quad x \geq 0,
\end{align*}
\]

where \( x \in \mathbb{R}^n \) is the decision variable and \( c^T \) denotes the coefficients of the objective function. \( A \in \mathbb{R}^{m \times n} \) is a random matrix and represents the coefficients and \( b \in \mathbb{R}^m \) is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability \( \epsilon = 1 - p \), where \( 0 < p \leq 1 \). Note that \( \epsilon \) is sometimes called the risk tolerance.

One way to solve a stochastic problem with chance constraints is to convert it to a mixed-integer problem (MIP) first and then solve the MIP equivalent. The idea is to introduce a set of scenarios \( S \) and a vector with binary variables, say \( y_k \in \mathbb{R}^m \), for each scenario \( k \in S \). The binary variables take value 1 if the corresponding constraint is satisfied in a scenario and 0 otherwise. A scenario-based formulation of the chance-constrained stochastic linear program above can be written as:

\[
\begin{align*}
\text{Min} & \quad c^T x \\
\text{s.t.} & \quad A^k x \leq b^k + M^k (1 - y_k) \\
& \quad \sum_{k \in S} y_k \geq p \times |S| \\
& \quad x \geq 0, \quad y \in (0, 1)^{|S|},
\end{align*}
\]

where \( M^k \in \mathbb{R}^m \) is a chosen big-M vector. The entries of the vector \( M^k \) should be chosen such that it does not cut off any feasible solution if \( y_k = 0 \).

4.51.11.3.2 Example with a Single Chance Constraint  

We start with the simplest case, where the random matrix \( A \) consists of just one random vector \( a \), resulting in a problem with a single chance constraint. The following example is adapted from model [SIMPLECHANCE].

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad P(\omega x_1 + x_2 \geq 7) \geq 0.75, \quad \omega \in \Omega = \{1, 2, 3, 4\}
\end{align*}
\]

Here the random parameter \( \omega \) has four possible realizations. Thus we have four scenarios and we assume that each scenario is equally likely to be realized, i.e. \( \pi^k = \frac{1}{4} \), where \( \pi^k \) denotes the probability that scenario \( k \) is realized. Note that in this example \( b \) is fixed at 7. As \( p = 0.75 \) and each scenario is equally likely to be realized, we need to choose \( x_1 \) and \( x_2 \) such that the inequality is satisfied in at least 3 scenarios. For clarity, we spell out the inequalities for the scenarios:

\[
\begin{align*}
& k = 1: \quad \omega_1 = 1 \quad \omega_1 x_1 + x_2 \geq 7 \\
& k = 2: \quad \omega_2 = 2 \quad \omega_2 x_1 + x_2 \geq 7 \\
& k = 3: \quad \omega_3 = 3 \quad \omega_3 x_1 + x_2 \geq 7 \\
& k = 4: \quad \omega_4 = 4 \quad \omega_4 x_1 + x_2 \geq 7.
\end{align*}
\]
The MIP equivalent of problem (49) is given below:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& \quad 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
& \quad 3x_1 + x_2 \geq 7 - M(1 - y_3) \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_4) \\
cc_1 &= 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 4, \quad \pi^k = \frac{1}{4} \\
x_1, x_2 &\geq 0 \\
0 \leq cc_1 \leq (1 - 0.75) \\
y_k &\in (0, 1).
\end{align*}
\] (51)

Observe that the first four constraints cover the four possible scenarios with \( \omega \) taking the values 1, 2, 3 and 4 respectively. On the right-hand side we introduce a big-M factor and \( y_k \), a binary indicator variable. \( y_k \) takes the value 1 if the constraint is satisfied and zero otherwise. The new variable \( cc_1 \) represents the probability that the constraint is violated. If \( cc_1 = 0 \), the sum will equal 1, indicating that the constraint is satisfied in all four scenarios. If \( cc_1 = 0.25 \), the constraint will be unsatisfied in one scenario out of four (for this scenario \( y_k = 0 \)).

The problem can be modeled in GAMS EMP as follows:

* Core Model
Positive Variables x1, x2;
Variables z;
Scalar om / 1 /;
Equations obj, e1;
obj.. z =e= x1 + x2;
e1.. om*x1 + x2 =g= 7;
Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp;
$onput
randvar om discrete 0.25 1 0.25 2 0.25 3 0.25 4
chance e1 0.75
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s4 /;
Parameter _s_om(scen)
x1_l (scen)
x2_l (scen)
e1_l (scen);
Set dict / scen .scenario.

om .randvar ._s_om
x1 .level .x1_l
x2 .level .x2_l
Extended Mathematical Programming (EMP)

\[ e_1 \ .level .e_1_l /; \]

solve sc min z use EMP scenario dict;
display s_om, x_1_l, x_2_l, e_1_l;

Like other EMP stochastic programming models, the model consists of three parts: the core model, the EMP annotations and the dictionary with output-handling information. As usual, the core model is defined as a deterministic model and the specifications relating to the stochastic structure of the problem are written to the file `emp.info`. In the EMP annotations, the EMP keyword `randvar` is used to declare a parameter from the core model as a random variable. The keyword is followed by `om`, the random variable, and the distribution. The EMP keyword `discrete` indicates that we have a discrete distribution. The discrete distribution is specified via probability-numerical value pairs. Note that continuous distributions are also possible. See section Random Variables with Continuous Distributions for more information.

The second line in the EMP annotations starts with the EMP keyword `chance` followed by the equation `e_1` and a probability. The specification means that the constraints `e_1` must hold for at least 75% of all scenarios. We can verify that this requirement has been enforced by checking in the listing file the level value of the constraint `e_1_l`. We will see that in the first scenario the constraint was violated, but it was satisfied in all other scenarios.

Observe that the EMP keyword `chance` allows for an optional specification that is related to a penalization factor for constraints that are violated. This topic is discussed in section Penalizing Violations of Chance Constraints below.

Note

Currently, there are no stages in stochastic problems with chance constraints, unlike in stochastic problems with recourse.

As the problem is solved as a MIP, it is important that the options OptCA and OptCR are assigned appropriate values.

Note that the default value of \( M \) is 1e7 for the solver Lindo. To customize this and set it to, e.g., 1000 instead, one could insert the following five lines before the solve statement:

```plaintext
option emp = lindo;
$onecho > lindo.opt
STOC_BIGM 1e3
$offecho
sc.optfile = 1;
```

For DE the default value of \( M \) is 10000 and it could be changed to 1000 in a similar way as seen above for Lindo:

```plaintext
option emp = de;
$onecho > de.opt
ccreform bigM 1e3
$offecho
sc.optfile = 1;
```

Note that `ccreform` is a DE option. This option allows to specify two alternative solution strategies: a reformulation using a convex hull and a reformulation using indicator variables and indicator constraints. The following line indicates that a convex hull with \( M = 1000 \) and \( \epsilon = 0.00001 \) is to be used:

```plaintext
ccreform cHull 1e3 1e-6
```

For indicator variables and constraints, the following line is used:

```plaintext
ccreform indic
```

Note that currently only the solver CPLEX supports indicator variables, so the resulting reformulated problem has to be solved with CPLEX.
4.51.11.3.3 Examples with Multiple Chance Constraints  Stochastic problems with multiple chance constraints have two different forms: problems with joint chance constraints and problems with multiple individual chance constraints. In problems with joint chance constraints the constraints all have to be satisfied simultaneously with probability \( p \). In problems with multiple individual chance constraints each constraint carries its own risk tolerance. We illustrate with two examples.

4.51.11.3.3.1 Joint Chance Constraints  We illustrate joint chance constraints by extending the example above by one constraint resulting in a problem with two chance constraints:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
s.t. & \quad P(\omega_1 x_1 + x_2 \geq 7; \omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad (\omega_1, \omega_2) \in \Omega \\
& \quad x_1, x_2 \geq 0,
\end{align*}
\]

where

\[
\Omega = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), (4,1), (4,2), (4,3)\}
\]

and

\[
\pi^k = \pi(\omega_1, \omega_2) = \frac{1}{12} \quad \text{for all} \quad (\omega_1, \omega_2) \in \Omega.
\]

Note that both random variables follow discrete uniform distributions, \( \omega_1 \) has four realizations and \( \omega_2 \) has three realizations. Thus we have 12 scenarios that are all equally likely. The MIP equivalent takes the following form:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
s.t. & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
& \quad 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_2) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
& \quad 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
& \quad cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad x_1, x_2 \geq 0 \\
& \quad 0 \leq cc_1 \leq (1 - 0.6) \\
& \quad y_k \in (0, 1).
\end{align*}
\]

Note that the first set of constraints covers the 12 scenarios, where each scenario has two constraints. The other constraints are similar to those introduced in the previous model.

The problem can be modeled with GAMS EMP as follows:

* Core Model
Positive Variables x1, x2;
Variables z;
Scalars om1, om2;
om1 = 1;
om2 = 1;
Equations obj, e1, e2;

obj.. z =e= x1 + x2;
e1.. om1*x1 + x2 =g= 7;
e2.. om2*x1 + 3*x2 =g= 12;

Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 e2 0.6
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s12 /;
Parameters s_om1(scen)
s_om2(scen)
x1_l (scen)
x2_l (scen)
e1_l (scen)
e2_l (scen);
Set dict / scen .scenario.''
om1 .randvar .s_om1
om2 .randvar .s_om2
x1 .level .x1_l
x2 .level .x2_l
e1 .level .e1_l
e2 .level .e2_l/;
solve sc min z use emp scenario dict;
display s_om1, s_om2, x1_l, x2_l, e1_l, e2_l;

In the EMP annotations, the line chance e1 e2 0.6 specifies that in at least 60% of all scenarios both constraints e1 and e2 must be satisfied at the same time. There are a total of 12 scenarios, hence both constraints must be satisfied in at least 8 scenarios (i.e. $12 \times 0.6 = 7.2 \leq 8$). We can verify that this requirement has been enforced by checking in the listing file the level values of the constraints, e1_l and e2_l. Indeed, in the optimal solution both constraints hold in scenarios 4 to 12, meaning that there are 9 scenarios that satisfy both inequalities.

4.51.11.3.3.2 Individual Chance Constraints In stochastic problems with individual chance constraints there is no correlation between the probabilities of the rows of the matrix $A$. Hence problem (47) takes the following form:

$$
\begin{align*}
\text{Min}_x & \quad c^T x \\
\text{s.t.} & \quad P(A_i x \leq b_i) \geq p_i, \quad i = 1, \ldots, m \\
x & \geq 0.
\end{align*}
$$

(54)
To illustrate, we use an extended version of the example from the previous section and replace the joint chance constraints by individual chance constraints:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad P(\omega_1 x_1 + x_2 \geq 7) \geq 0.75, \quad \omega_1 \in \Omega_1 = \{1, 2, 3, 4\} \\
& \quad P(\omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad \omega_2 \in \Omega_2 = \{1, 2, 3\} \\
& \quad P(\omega_1 x_1 + \omega_2 x_2 \geq 10) \geq 0.5, \quad (\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 = \Omega \\
& \quad x_1, x_2 \geq 0.
\end{align*}
\]

Note that \(\Omega\) is defined as in (52) above, we have again 12 scenarios, each with probability \(\pi^k = \frac{1}{12}\). However, in this example, the first inequality must hold in 9 out of 12 scenarios (0.75 \times 12 = 9), the second inequality must hold in 8 out of 12 inequalities (0.6 \times 12 = 7.2 \leq 8) and the third inequality must hold in 6 out of 12 scenarios. Note further, that we may have four types of scenarios: scenarios where all constraints are violated, scenarios where two constraints are violated, scenarios where one constraint is violated and scenarios where all three constraints are satisfied. The only condition is that for each constraint there is the respective number of scenarios where the constraint is satisfied. Observe that the random data in the third inequality is a combination of the random data of the first two inequalities. The inequalities for the scenarios are given below:

\[
\begin{align*}
k = 1: & \quad \omega_1^1 = 1, \ \omega_2^1 = 1 \quad \omega_1^1 x_1 + x_2 \geq 7 \\
& \quad \omega_2^1 x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^1 x_1 + \omega_2^1 x_2 \geq 10 \\
k = 2: & \quad \omega_1^2 = 1, \ \omega_2^2 = 2 \quad \omega_1^2 x_1 + x_2 \geq 7 \\
& \quad \omega_2^2 x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^2 x_1 + \omega_2^2 x_2 \geq 10 \\
& \quad \vdots \\
k = 12: & \quad \omega_1^{12} = 4, \ \omega_2^{12} = 3 \quad \omega_1^{12} x_1 + x_2 \geq 7 \\
& \quad \omega_2^{12} x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^{12} x_1 + \omega_2^{12} x_2 \geq 10
\end{align*}
\]

The MIP equivalent follows:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1^1) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_1^2) \\
& \quad 1x_1 + 1x_2 \geq 10 - M(1 - y_1^3) \\
& \quad 2x_1 + x_2 \geq 7 - M(1 - y_1^4) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_1^5) \\
& \quad 2x_1 + 1x_2 \geq 10 - M(1 - y_1^6) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_{12}^1) \\
& \quad 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}^2) \\
& \quad 4x_1 + 3x_2 \geq 10 - M(1 - y_{12}^3) \\
& \quad cc_1 = 1 - \sum_{k=1}^{12} \pi^k y_k^1, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad cc_2 = 1 - \sum_{k=1}^{12} \pi^k y_k^2, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad cc_3 = 1 - \sum_{k=1}^{12} \pi^k y_k^3, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad x_1, x_2 \geq 0 \\
& \quad 0 \leq cc_1 \leq (1 - 0.75) \\
& \quad 0 \leq cc_2 \leq (1 - 0.6) \\
& \quad 0 \leq cc_3 \leq (1 - 0.5) \\
& \quad y_k^i \in (0, 1).
\end{align*}
\]
As expected, there are three constraints for every scenario. Note that we introduced three new variables, \( cc_1 \), \( cc_2 \) and \( cc_3 \) and three corresponding constraints. Each of the variables has a different range reflecting the different probabilities with which a constraint may be violated.

The core model of the GAMS code is very similar to the core model with joint chance constraints. We just add the third inequality:

Equations obj, e1, e2, e3;

\[
\begin{align*}
\text{obj..} & \quad z =e= x_1 + x_2; \\
\text{e1..} & \quad \text{om}_1 x_1 + x_2 \geq 7; \\
\text{e2..} & \quad \text{om}_2 x_1 + 3x_2 \geq 12; \\
\text{e3..} & \quad \text{om}_1 x_1 + \text{om}_2 x_2 \geq 10;
\end{align*}
\]

There is a slight modification in the annotations:

File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 0.75
chance e2 0.6
chance e3 0.5
$offput
putclose emp;

Observe that we have three lines that start with the EMP keyword chance: every constraint is listed separately with its respective probability.

Note
In case one constraint has to be satisfied in all scenarios (so it is strictly speaking not a chance constraint), then it has to be listed with probability 1.0.

An overview of which constraints are satisfied in which scenarios in the optimal solution is given in the following table:

<table>
<thead>
<tr>
<th>Scenarios Where</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1 is satisfied</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios where e1 is satisfied</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios where e1 is satisfied</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios where e1 is satisfied</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Overview of constraint satisfaction per scenario

Observe that all constraints are satisfied in as many scenarios as required and there are scenarios where all three constraints are satisfied (\( k = 6, 8, 9, 11, 12 \)), scenarios where only two constraints are satisfied (\( k = 5, 7, 10 \), scenarios where only one constraint is satisfied (\( k = 2, 3, 4 \)) and one scenario where all constraints are violated (\( k = 1 \)).

The choice whether joint or individual chance constraints should be used depends on the system being modeled. Both approaches have their own advantages. Individual chance constraints are weaker since not
all constraints have to be satisfied at the same time. This can be clearly observed in the optimal solution for example (55). The objective value is 5.20 in the model with joint chance constraints and 4.75 in the model with individual chance constraints (assuming that each constraint is satisfied in 60% of all scenarios). As it is a minimizing problem, the model with individual chance constraints yields the better result. However, in this solution we have only 6 scenarios where both constraints are simultaneously satisfied while each constraint is satisfied in eight scenarios in total for the joint chance constraint formulation, as required.

4.5.11.3.4 Penalizing Violations of Chance Constraints The EMP framework provides the syntax for a penalty factor for each scenario that violates one or more constraints. In the example with joint constraints above, the EMP annotations could be modified in the following way:

```
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 e2 0.6 3
$offput
```

Note that we added the number 3 at the end of the last line. This new entry represents a penalty factor. Recall that the MIP equivalent (53) takes the following form:

\[
\begin{align*}
\text{Min} & \quad z = x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
& \quad 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
& \quad cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad x_1, x_2 \geq 0 \\
& \quad 0 \leq cc_1 \leq (1 - 0.6) \\
& \quad y_k \in (0, 1).
\end{align*}
\]  

(58)

Observe that the probability with which the constraints are allowed to be violated is stored in the variable \(cc_1\). Introducing a penalty factor or weight \(w\) has the effect that the term \((w \times cc_1)\) is added to the objective function:

\[
z = x_1 + x_2 + w \times cc_1 = x_1 + x_2 + 3 cc_1
\]

(4.29)

Similarly, we could add weights in the example with individual chance constraints:

```
chance E1 0.75 5
chance E2 0.6 6
chance E3 0.5 7
```

Given these annotations, we obtain the following objective function in the MIP equivalent:

\[
z = x_1 + x_2 + 5 cc_1 + 6 cc_2 + 7 cc_3.
\]

(4.30)

Penalty terms can be useful to explore sensitivities to slight changes.
Note

Like in the context of soft constraints, the penalty term is added in minimization problems and subtracted in maximization problems.

Moreover, the probability expression $cc_1$ in the MIP equivalent may be used as a variable in the original model. For example, in the joint chance constraints problem above, we could introduce a new variable, $viol$, in the objective function:

$$\text{Min } z = x_1 + x_2 + 3 * viol$$  \hspace{1cm} (4.31)

Then we write the chance constraint specification in the EMP annotations as follows:

```
chance E1 E2 0.6 viol
```

The addition of the variable $viol$ as penalty factor causes $cc_1$ to be replaced by $viol$ in the MIP equivalent. Thus we have:

$$viol = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12}, \quad viol \in [0, 0.4].$$  \hspace{1cm} (4.32)

Note that the corresponding model is equivalent to the joint chance constraints model with penalty factor 3 with which we started this subsection.

### 4.51.11.3.5 EMP Syntax for Chance Constraints

The general syntax of the EMP annotations used to specify stochastic problems with chance constraints is as follows:

```plaintext
[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val {val} {prob val val {val}}]
chance equ {equ} [holds] minRatio [weight|var]
{chance equ {equ} [holds] minRatio [weight|var]}
```

The first three lines present three ways to specify random variables. The syntax and logic is the same as in stochastic problems with recourse, see above for details. Note that at least one random variable has to be specified. The EMP keyword `chance` is used to define the constraint(s) that only have to hold for a certain ratio of the possible outcomes. This ratio is given with the number `minRatio` ($0 \leq \text{minRatio} \leq 1$). The keyword `holds` is optional and does not affect the solver. The remaining specifications are optional and relate to the penalization of chance constraints that are violated. If `weight` is defined, the violation of a chance constraint will be penalized in the objective function ($\text{weight} \times \text{violationRatio}$). Alternatively, if the name of a variable of the model (`var`) is specified, the violation will be multiplied by the value of this variable instead of the fixed multiplier `weight`.

In this section we have introduced various versions of problems with chance constraints with simple examples. For a more complex example, see the farming model `[KILOSAFARM]` in the GAMS EMP Model Library.

### 4.51.12 EMP Keywords

Some words act as keywords in the context of the EMP annotations in the file `emp.info`, but they are not GAMS reserved words, i.e. they are not keywords in the GAMS code apart from the EMP annotations. In this section we present an overview of all GAMS EMP keywords, ordered according to the type of programming where they are used.

#### 4.51.12.1 GAMS EMP Keywords for Soft Constraints
### 4.51.12.2 GAMS EMP Keywords for Variational Inequalities

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI</td>
<td>Indicates that the specifications that follow are a variational inequality. See the discussion of EMP specifications for VI and the example above.</td>
</tr>
<tr>
<td>QVI</td>
<td>Indicates that the specifications that follow are a quasi-variational inequality. See the discussion of EMP specifications for QVI and the example above.</td>
</tr>
</tbody>
</table>

### 4.51.12.3 GAMS EMP Keywords for Equilibrium Problems

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dualvar</td>
<td>See below.</td>
</tr>
<tr>
<td>Equilibrium</td>
<td>Indicates that the specifications that follow define the structure of an equilibrium problem. For examples, see sections Equilibrium Problems with EMP: A Simple Example and Equilibrium Problems with EMP: Example with Dual Variables. The general syntax of EMP annotations for equilibrium problems is introduced in section EMP Syntax for Equilibrium Problems.</td>
</tr>
<tr>
<td>implicit</td>
<td>Specifies a shared variable and its defining constraint. For details, see section Equilibrium Problems with Shared Variables.</td>
</tr>
<tr>
<td>max</td>
<td>This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the maximization problem of one agent.</td>
</tr>
<tr>
<td>min</td>
<td>This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the minimization problem of one agent.</td>
</tr>
<tr>
<td>VI</td>
<td>Indicates that the specifications that follow define a VI, as in this example.</td>
</tr>
<tr>
<td>VIsol</td>
<td>Specifies that the equation that follows is a shared constraint and prompts the EMP framework to use the MCP reformulation where a variational inequality is associated with the shared constraint. For more information and an example, see section Equilibrium Problems with Shared Constraints.</td>
</tr>
</tbody>
</table>

### 4.51.12.4 GAMS EMP Keywords for Embedded Complementarity Systems

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DualEqu</td>
<td>This keyword is followed by an equation-variable pair. It establishes a complementarity relationship between an external equation and the named variable of the model. An example is given above. See also the spatial price equilibrium model [HARK-MONOP].</td>
</tr>
<tr>
<td>DualVar</td>
<td>This keyword is followed by a variable-equation pair. It specifies that the variable is the dual of the equation. Examples are discussed in sections Equilibrium Problems with EMP: Example with Dual Variables and Embedded Complementarity Systems.</td>
</tr>
</tbody>
</table>
Note that problems with embedded complementarity systems can be recast as equilibrium problems. Details are given in section Embedded Complementarity Systems.

### 4.51.12.5 GAMS EMP Keywords for Bilevel Programming

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bilevel</td>
<td>Indicates that the specifications that follow relate to a bilevel programming problem. The keyword is followed by the decision variable(s) of the upper-level problem and the definition(s) of the lower-level problem(s). For examples and more details, see section Bilevel Programs.</td>
</tr>
<tr>
<td>dualvar</td>
<td>See above.</td>
</tr>
<tr>
<td>max</td>
<td>This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the maximization problem of one agent.</td>
</tr>
<tr>
<td>min</td>
<td>This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the minimization problem of one agent, as in this example.</td>
</tr>
<tr>
<td>VI</td>
<td>Indicates that the specifications that follow define a VI, as in this example.</td>
</tr>
</tbody>
</table>

### 4.51.12.6 GAMS EMP Keywords for Disjunctive Programming

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigM</td>
<td>Indicates that the big M reformulation method shall be used.</td>
</tr>
<tr>
<td>chull</td>
<td>Indicates that a convex hull shall be used for the reformulation. Note that this is the default method.</td>
</tr>
<tr>
<td>default</td>
<td>Indicates that the specification that follows is the reformulation method.</td>
</tr>
<tr>
<td>disjunction</td>
<td>Indicates that the specification that follows is a disjunction. For a discussion of the general syntax following this keyword, see above.</td>
</tr>
<tr>
<td>indic</td>
<td>Indicates that indicator constraints shall be used as reformulation method.</td>
</tr>
<tr>
<td>star (*)</td>
<td>The symbol * will be replaced by internal default binary variables, thus explicit binary variables that model the Boolean variables are not needed. Note that default binary variables may only be used if there are no logic equations in the model. For an example and more details, see above.</td>
</tr>
</tbody>
</table>

### 4.51.12.7 GAMS EMP Keywords for Stochastic Programming

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chance</td>
<td>Defines individual or joint chance constraints.</td>
</tr>
<tr>
<td>cvar</td>
<td>Synonym to cvarup.</td>
</tr>
<tr>
<td>cvarlo</td>
<td>This keyword assigns a variable to have the value $\text{CVaR}_\alpha$, where $\alpha$ is a scalar that represents the confidence level for the Conditional Value at Risk. Note that cvarlo refers to the left tail of the distribution. For more details and examples, see section Conditional Value at Risk (CVaR).</td>
</tr>
<tr>
<td>cvarup</td>
<td>This keyword assigns a variable to have the value $\text{CVaR}_\alpha$. $\alpha$ is a scalar that represents the confidence level for the Conditional Value at Risk. Note that cvarup refers to the right tail of the distribution.</td>
</tr>
<tr>
<td>discrete</td>
<td>Indicates that the specification that follows is the discrete distribution of one or more random variables.</td>
</tr>
<tr>
<td>ExpectedValue</td>
<td>This keyword is used to state that a variable is the expected value of a random variable.</td>
</tr>
<tr>
<td>EMP Keyword</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>jrandvar</td>
<td>This keyword is used to define discrete random variables that are jointly distributed. At least two random variables must be named. For an example, see the news vendor model ([\text{NBDISCJOINT}]).</td>
</tr>
<tr>
<td>randvar</td>
<td>This keyword declares that a parameter of the model is in fact a stochastic random variable. The keyword is followed by the name of the parameter and details about the probability distribution (discrete or parametric). A list of all supported parametric distributions is given in Table 4.</td>
</tr>
<tr>
<td>sample</td>
<td>This keyword allows users to customize the size of the sample of one or more random variables from a continuous distribution and - optionally - to determine the variance reduction method to be used. An example and further details are given above. Note that without a valid LINDO license this is limited to the Normal and Binomial distributions with a maximum sample size of 10.</td>
</tr>
<tr>
<td>setSeed</td>
<td>This keyword sets the seed for the random number generator of the sampling routines that are called via the keyword sample. If setSeed is used in EMP annotations, the seed will be set once before all samples will be generated. Note that a valid LINDO license is required to use this keyword.</td>
</tr>
<tr>
<td>stage</td>
<td>This keyword is followed by a number and the names of the random variables, variables and equations that are assigned to the respective stage. Stage 1 is the default stage for all random variables, variables and equations that are not assigned a stage explicitly except for objective equation and variable. Their default stage is the highest stage in the model.</td>
</tr>
<tr>
<td>stageDefault</td>
<td>This keyword is followed by a number. This specifies the default stage for all random variables, variables and equations that are not assigned a stage explicitly (except for objective equation and variable). If this keyword is not used explicitly, the default is 1.</td>
</tr>
<tr>
<td>var</td>
<td>Synonym to varup.</td>
</tr>
<tr>
<td>varlo</td>
<td>This keyword assigns a variable to have the value (\text{VaR}_\alpha), where (\alpha) is a scalar that represents the percentile of the Value at Risk. Note that varlo refers to the left tail of the distribution. For more details and examples, see section Value at Risk (VaR).</td>
</tr>
<tr>
<td>varup</td>
<td>This keyword assigns a variable to have the value (\text{VaR}_\alpha), where (\alpha) is a scalar that represents the percentile of the Value at Risk. Note that varup refers to the right tail of the distribution and (\alpha) typically equals 0.95 or 0.9. For further information, see section Value at Risk (VaR).</td>
</tr>
</tbody>
</table>

Currently, three GAMS solvers can be used to solve stochastic programming models with EMP: DE, DECIS, and LINDO. Not all keywords mentioned above are supported by all those solvers. The following table specifies which keywords can be used with which solvers. The keywords not mentioned in the table are supported by all solvers mentioned.

<table>
<thead>
<tr>
<th>EMP Keyword</th>
<th>DE</th>
<th>DECIS</th>
<th>LINDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>chance</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>jrandvar</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>randvar</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>randvar(\text{discrete})</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>randvar(\text{parametric})</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>sample</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>setSeed</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>var</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvar</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExpectedValue</td>
<td>√</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.52 Accessing Model Libraries

Table 8: Solver capabilities

Note

In general, JAMS is the default solver for models of type EMP. However, since JAMS cannot handle stochastic EMP models, GAMS switches the solver to DE automatically when a stochastic EMP model is tried to be solved with JAMS.

Further information about these solvers can be found in the corresponding solver manuals. The stochastic programming options for the solver LINDO might be of particular interest.

4.52 Accessing Model Libraries

Professor Paul Samuelson is fond of saying that he hopes each generation of economists will be able to "stand on the shoulders" of the previous generation. The library of models included with the GAMS system is a reflection of this desire. We believe that the quality of modeling will be greatly improved and the productivity of modelers enhanced if each generation can stand on the shoulders of the previous generation by beginning with the previous models and enhancing and improving them. The Model Libraries includes a large number of models, collectively organized as:

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.

- **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.

- **GAMS Data Utilities Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.

- **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.

- **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.


- **NOA Library** - includes GAMS nonlinear optimization applications models based on the book *Nonlinear Optimization Applications Using the GAMS Technology* by Neculai Andrei.

- **PSOPT Library** - includes GAMS optimization models based on the book *Power System Optimization Modelling in GAMS* by Alireza Soroudi.

The models included have been selected not only because they collectively provide strong shoulders for new users to stand on, but also because they represent interesting and sometimes classic problems. For example the trade-off between consumption and investment is richly illustrated in the Ramsey problem, which can be solved using nonlinear programming methods. Examples of other problems included in the library are production and shipment by firms, investment planning in time and space, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and relational databases.
Another criterion for including models in the library is that they illustrate the modeling capabilities GAMS offers. For example, the mathematical specification of cropping patterns can be represented handily in GAMS. Another example of the system's capability is the style for specifying initial solutions as staring points in the search for the optimal solution of dynamic nonlinear optimization problems.

Finally, some models have been selected for inclusion because they have been used in other modeling systems. Examples are network problems and production planning models. These models permit the user to compare how problems are set up and solved in different modeling systems.

Most of the models have been contributed by GAMS users. The submission of new models is encouraged. If you would like to see your model in a future release of the library, please send the model and associated documents and reports to GAMS Development Corporation.

4.52.1 Usage

4.52.1.1 Command Line Approach

One way to access the library is through command line. The following commands copy a model from the library directory into the current directory.

<table>
<thead>
<tr>
<th>Command</th>
<th>Library to access</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamslib</td>
<td>GAMS Model Library</td>
</tr>
<tr>
<td>testlib</td>
<td>GAMS Test Library</td>
</tr>
<tr>
<td>datalib</td>
<td>GAMS Data Library</td>
</tr>
<tr>
<td>emplib</td>
<td>GAMS EMP Library</td>
</tr>
<tr>
<td>apilib</td>
<td>GAMS API Library</td>
</tr>
<tr>
<td>finlib</td>
<td>FIN Library</td>
</tr>
<tr>
<td>noalib</td>
<td>NOA Library</td>
</tr>
<tr>
<td>psoptlib</td>
<td>PSOPT Library</td>
</tr>
</tbody>
</table>

Take the command gamslib for an example, if you enter gamslib without any parameters, the command syntax will be displayed as shown below:

> gamslib modelname [target]

or

> gamslib modelnum [target]

where

- modelname is the modelname
- modelnum is the model sequence number, and
- target is the target file name.

If the target file name is not provided, the default is modelname.gms. The file will be automatically copied into the current working directory. For example, the [TRNSPORT] model from the GAMS Model library has sequence number 1 and could be copied in any of the following ways.

To copy [TRNSPORT] model file to target file trnsport.gms under the current directory:
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE)

> gamslib trnsport

> gamslib 1

To copy [TRNSPORT] model file to target file myname.gms under the current directory:

> gamslib trnsport myname

> gamslib 1 myname

The other commands have similar usage to gamslib command.

4.52.1.2 IDE Approach

A convenient way to access the model library on the major platforms is the "Model Library Explorer" from GAMS Studio. It is described in detail in the GAMS Studio documentation.

Users may define their own library by using a GLB file. These libraries can also be accessed through Studio.

4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE)

- Introduction to MPSGE
- MPSGE Models in GAMS
- Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE
- Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae
- A Library of Small Examples for Self-Study
- Linking Implan Social Accounts to MPSGE
- The MPSGE guide is also available as PDF.
4.53.1 Introduction to MPSGE

MPSGE is a language used for formulating and solving Arrow–Debreu economic equilibrium models and exists as a subsystem within GAMS. The name stands for ‘mathematical programming system for general equilibrium’. MPSGE provides a short-hand non-algebraic representation for the systems of nonlinear inequalities which underly general equilibrium models. The MPSGE framework is based on nested constant elasticity of substitution utility and production functions. The data requirements for a model include share and elasticity parameters, endowments, and tax rates for all the consumers and production sectors in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

The main benefit of using MPSGE is that modelers are released from having to write the equations of the model and the calibrated demand and supply functions. Experience shows that even computing the parameters of the functions is error-prone. The tabular input format of MPSGE facilitates a compact, non-algebraic representation of the equations of a model. Thus algebraic tedium and the scope for programming errors are reduced, and more attention can be paid to economic interpretation and testing of alternative formulations.

This chapter offers an introduction to MPSGE for users who are familiar with GAMS. For more information on MPSGE and details on more advanced features, see [156] and [130]. In addition, a library of small examples for self-study is available here.

This chapter is organized as follows. We start with an overview of some basic economic ideas and general remarks on specifying an MPSGE model. Next, we introduce a first simple example. Then we demonstrate how intermediate demand and joint production are modeled with MPSGE. We follow with a third model that features a small open economy and also includes taxes on inputs and outputs, labor-leisure choice, classical unemployment, and nested demand functions. We conclude the chapter with an overview of MPSGE keywords and the syntax for specifying functions, some other features like domain restrictions on variable declarations, and a few brief comments on MPSGE-specific output in the GAMS listing file. Note that a full version of all MPSGE models that are discussed, including standard GAMS versions, are given in the Appendix.

4.53.1.1 Some Basic Economic Ideas

MPSGE models are based on the following economic ideas. In each model there are multiple interacting agents, where each agent solves an optimization problem. For example, consumers maximize utility subject to their budget constraint, and producers minimize their cost given available technology. Agent interactions are mediated by markets and prices and the optimal solution is an equilibrium where prices and activity levels have adjusted so that markets for produced goods and input factors clear. In equilibrium, each agent cannot do better by altering their behavior (given the constraints they face). In this section we give more details on these basic economic ideas. In particular, we offer brief introductions to consumer demand theory, producer supply theory and general economic equilibrium.

4.53.1.1.1 Consumer Demand Theory

Consumer choice is modeled using an optimization paradigm: consumers form a consumption bundle which maximizes welfare subject to a budget constraint. As an abstract notion, this seems plausible, but certain details must be dealt with to provide an operational application of the idea.

Consider the following example. Thomas lives in Ann Arbor where he spends 30% on housing and the rest of his income on other goods. This information is essentially an observation of a benchmark equilibrium, consisting of the prevailing prices and quantities of goods demanded. Now, Thomas has an employment offer in Berlin which pays 50% more than he currently earns, but he is hesitant to take the job since rental rates in Berlin are three times higher than in Ann Arbor. The question is: On purely economic grounds, should he move?
Thomas’ choices are illustrated below. Point $a$ represents the benchmark point, where he spends the amount $H_a$ on housing in Ann Arbor. Point $b$ represents a potential trade-off between housing and other goods in Berlin, where he spends a lower percentage of his income on housing and more on other goods.

![Figure 4.4 Thomas’ Choices](image)

Whether Thomas derives the same welfare from the choices represented by point $b$ like those represented by point $a$ depends on the elasticity of substitution. The elasticity of substitution is the willingness to exchange some amount of one good for more of the other good. The three curves in the figure above represent three elasticities of substitution. Note that the less convex (flatter) the curve is, the higher is the elasticity, i.e. the more willing is Thomas to substitute housing for other goods.

If the elasticity of substitution equals $\sigma^*$, Thomas is indifferent between choices $a$ and $b$, since both points lie on the respective indifference curve. However, if the elasticity of substitution is lower, say equals $\sigma_L$, or higher, say it equals $\sigma_H$, then $b$ will be worse than $a$ and Thomas will not choose it. Hence, the elasticity of substitution is crucial in determining whether Thomas should move on economic grounds.

The utility function can be deduced if prices and the associated choices are observed and the elasticity of substitution is given. Thus, in MPSGE, a utility function is represented by benchmark (or observed) quantities, benchmark prices and the associated elasticity of substitution. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point. The elasticity of substitution is a measure of the curvature of the indifference curve and specifies the utility function unambiguously.

### 4.53.1.1.2 Producer Supply Theory

While consumers aim to maximize their utility subject to a budget, producers aim to reduce their cost of production given available technology. The activity of a firm is modeled as a production function that maps inputs into outputs. Inputs are primary factors like labor, capital, land, raw materials etc. and outputs are goods and services. Similarly to utility functions, in MPSGE, a production function is represented by reference quantities of reference prices of inputs and outputs, elasticities of substitution and elasticities of transformation.

Summarizing, the main principle for both, consumers and producers, is choice. Consumers choose quantities of goods given their preferences, prices and budgets. Producers choose quantities of primary factors and output levels given available technology. Available technology is modeled via elasticities of substitution between inputs and elasticities of transformation between outputs.
4.53.1.1.3 General Economic Equilibrium  As mentioned above, MPSGE is a system for modeling general economic equilibria. General economic equilibrium analysis studies the working of an economic system as a whole. The basic assumption is that there are two types of agents in the economy: producers and consumers. Producers are assumed to operate with constant return to scale and to be perfectly competitive price-taking firms. Further, they are assumed to select factor inputs and output levels in order to maximize their profits given available technology. Consumers are assumed to maximize their welfare subject to budget constraints (incomes and endowments). An economy is in economic equilibrium if all consumers spend their given incomes such that they gain maximum satisfaction, all firms in each sector minimize their costs given available technology, and prices (for primary factors and commodities) and activity levels have adjusted such that all agents cannot do better by altering their behavior given the constraints they face. In equilibrium, all markets have cleared: in both, goods and factors markets, total demand equals total supply.

Mathematically, such an economy is modeled as a system of weak inequalities, where each inequality is associated with a non-negative variable denoting a price or a quantity. If a particular weak inequality holds as an equation, then the associated variable is strictly positive. If it holds as a strict inequality, then the associated variable equals zero. This is a mixed complementarity problem that can be formulated and solved with standard GAMS syntax. For details see section Mixed Complementarity Problems (MCPs). As we will demonstrate, MPSGE models can be translated to standard GAMS MCP models. However, the framework that MPSGE offers is much more user-friendly, since modelers do not have to input the complex equations manually.

4.53.1.2 Specifying an MPSGE Model

In this section we give an overview of the MPSGE model specification, a simple introductory example follows in the next section. Like any standard GAMS program, a program with an embedded MPSGE model begins with set and parameter definitions that will later be used in the model. The MPSGE model is specified within an $ontext / $offtext block.

First, the name of the model must be defined:

\$MODEL:mymodel

Here $MODEL is an MPSGE keyword and the model name mymodel is a GAMS identifier.

In a GAMS program lines within $on/offtext are are treated as a block comment. However, if the preprocessor encounters the MPSGE model definition statement, it will recognize that an MPSGE model specification follows and will process the code accordingly.

Note

Like GAMS, MPSGE is not case sensitive.

Secondly, the variables for an MPSGE model are declared. In every model, there are three classes central variables:

- $SECTORS: variables for activity levels associated with constant returns to scale production sectors in the economy, which are non-negative,
- $COMMODITIES: variables for commodity prices including all final goods, intermediate goods and primary factors of production, which are also non-negative, and
- $CONSUMERS: variables for income levels, one for each “household” in the model, including any government entities.

In addition, models may include auxiliary variables ($AUXILIARY) that are used to introduce endogenous tax instruments or endogenous endowment quantities to a model.
Note

Unlike GAMS models, variables in an MPSGE model must be declared over an explicit domain. Only variables that have been declared as one of the four MPSGE variable types may be used in the MPSGE model. Error messages are generated for variables which are declared but not referenced.

Thirdly, the parameters for functions are specified in a structured manner. The general syntax for all possible entries is given in section Syntax for Production and Demand Functions below. The MPSGE framework uses these specifications to automatically generate the respective market-clearing and income-balance equations. There are two types of functions:

- **$PROD**: This block defines a production function, specifying inputs, outputs and elasticities of substitution and transformation. A production function must be given for each sector in the model. Note that most of the power and subtleties of the MPSGE framework center on the $PROD tables.

- **$DEMAND**: This block defines a demand function. This function represents preferences (using reference demands), initial factor endowments and elasticities of substitution. A demand function must be specified for each consumer in the model.

Finally, any auxiliary variable has an associated equilibrium condition that is defined by the user with conventional GAMS algebraic syntax and placed in a block called $CONSTR.

These four parts - model definition, variable declarations, function specifications and possibly side constraints for auxiliary variables - are the core of the MPSGE model specification. The model specification is followed by the dollar control option $offtext and the following compiler directive:

```
$sysinclude mpsgeset mymodel
```

This compiler directive instructs MPSGE to compile the functions and generates an external file called MYMODEL.GEN. This external file is then loaded to the GAMS file with the following directive:

```
$include MYMODEL.GEN
```

Finally, a standard GAMS solve statement follows and the model will be handed for solution to one of the MCP solvers:

```
solve mymodel using mcp;
```

Observe that the appropriate GAMS model type is MCP.

Most often the model is first evaluated at the benchmark point to test whether it is properly calibrated and everything works as expected. This is achieved by setting the option iterlim to zero. Then one or more counterfactuals are solved: first some parameter is modified using standard GAMS syntax, then the $include statement is repeated and the model is solved again.
4.53.1.3 An Introductory Example: Two Goods and Two Factors in a Closed Economy

In the canonical 2x2 model, two final goods, X and Y, are produced with two primary factors, labor L and capital K, and there is a single representative agent (consumer) RA. The model is defined by technology, preferences and endowments. We can describe preferences by a utility function which provides an ordinal ranking of consumption levels for X and Y. We assume that both factors are in fixed supply, thus in equilibrium, factor endowments equal factor demands, and the supply of the two goods equal their demand.

The problem can be cast as consisting of three production activities, X, Y and U, and five markets X, Y, U, K and L. The initial data for the model are given by the following matrix.

<table>
<thead>
<tr>
<th>Production Sectors</th>
<th>Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Markets</td>
<td>X</td>
</tr>
<tr>
<td>---------------------</td>
<td>----</td>
</tr>
<tr>
<td>PX</td>
<td>100</td>
</tr>
<tr>
<td>PY</td>
<td>50</td>
</tr>
<tr>
<td>PU</td>
<td>150</td>
</tr>
<tr>
<td>PK</td>
<td>-50</td>
</tr>
<tr>
<td>PL</td>
<td>-50</td>
</tr>
<tr>
<td>Column Sums</td>
<td>0</td>
</tr>
</tbody>
</table>

The data describes a representative equilibrium. Rows correspond to markets, where PX, PY, PU, PK and PL are the prices for the commodity X, the commodity Y, the utility U, and capital and labor respectively. Positive entries denote the value of commodity flows into the economy (sales or factor supplies) and negative entries represent the value of commodity flows out of the economy (factor demands and good demands). Observe that the sum of a row will equal zero if the market clears, i.e. the total amount of commodity that flows into the economy equals the total amount of this commodity that flows out of the economy. The matrix is called balanced if all rows and columns sum to zero.

Note that the layout of the matrix ensures that a complete list of the transactions associated with an activity is given in each production column. The sum of a production sector column is zero if the value of the outputs equals the costs of the inputs. A consumer column is balanced if the sum of primary factor sales equals the value of final demands. Hence columns that sum to zero indicate zero profits. This type of matrix is related to the concept of a social accounting matrix (or short: SAM).

The problem can be modeled with GAMS MPSGE as follows:

```plaintext
Parameter endow index of labour endowment / 1.0 /;

$ontext
$MODEL:twobytwo

$SECTORS:
  X  ! Activity level for sector X -- benchmark=1
  Y  ! Activity level for sector Y -- benchmark=1
  U  ! Activity level for sector U -- benchmark=1

$COMMODITIES:
  PX  ! Relative price index for commodity X -- benchmark=1
  PY  ! Relative price index for commodity Y -- benchmark=1
  PU  ! Relative price index for commodity U -- benchmark=1
```
PL ! Relative price index for labor -- benchmark=1
PK ! Relative price index for capital -- benchmark=1

$CONSUMERS:
   RA ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
   0:PX Q:100
   I:PL Q:50 ! Variable LX in the algebraic model
   I:PK Q:50 ! Variable KX in the algebraic model

$PROD:Y s:1
   0:PY Q:50
   I:PL Q:20 ! Variable LY in the algebraic model
   I:PK Q:30 ! Variable KY in the algebraic model

$PROD:U s:1
   0:PU Q:150
   I:PX Q:100 ! Variable DX in the algebraic model
   I:PY Q:50 ! Variable DY in the algebraic model

$DEMAND:RA
   D:PU
   E:PL Q: (70*endow)
   E:PK Q: 80

* Compiler directive instructing MPSGE to compile the functions
$sysinclude mpsgeset twobytwo

* Benchmark replication
twobytwo.iterlim = 0;
include TWOBYTWO.GEN
solve twobytwo using mcp;
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment
endow = 1.1;

* Solve the model with the default normalization of prices which
* fixes the income level of the representative agent. The RA
* income level at the initial prices equals 80 + 1.1*70 = 157.
include TWOBYTWO.GEN
solve twobytwo using mcp;

Note that an algebraic version of this model using standard GAMS syntax is given in the Appendix.

After the model statement where the model is named, variables are declared in each of the three central
variable classes. Unlike conventional GAMS syntax, in MPSGE, trailing comments in variable declarations
(starting with the symbol !) are interpreted as variable descriptors that will appear in the listing file.

Then three production blocks $PROD - one for each sector - follow. Output parameters are specified in
the line starting with the label O and input parameters are given in the line starting with the label I,
the respective quantities are listed in the field $Q$. Note that the first input in the production block for sector $X$ represents labor demand in sector $X$ and the second input represents capital demand in sector $X$. Similarly, the inputs in the production block for sector $Y$ represent labor and capital demand for sector $Y$. In the production block for sector $U$, the inputs represent demand for $X$ and $Y$ in sector $U$. In the field $s$: next to the name of the sector the elasticity of substitution between the inputs is specified.

A demand block $\$DEMAND$ for the consumer $RA$ completes the model specification. Demand parameters are specified in the line starting with the label $D$ and endowment parameters are given in the line starting with the label $E$. Note that all possible entries for production and demand blocks are discussed in section Syntax for Production and Demand Functions. Observe that there is no algebraic formulation of equations, only parameter specifications. Given this input, the MPSGE framework generates the respective equations automatically.

Note that it is not necessary to fix a numeraire. If a numeraire is not specified, the normalization of prices is arbitrary. For example, in the model above, $RA$ is used as a numeraire and the following remark appears in the listing file:

Default price normalization using income for $RA$

Note that the version of this model in the Appendix includes two additional counterfactual scenarios, where first sector $X$ is used as numeraire commodity and then the wage rate $PL$ is fixed as numeraire. In the Appendix, there are also two alternative versions of the model above: a model in standard GAMS syntax where the equations that are automatically generated by the MPSGE framework are given explicitly, and an MPSGE model that uses vector notation instead of scalars.

4.53.1.4 Modeling Intermediate Demand and Joint Production

In this section we extend the two by two model above in the following three ways:

1. We distinguish between production sectors and produced goods.
2. Each sector produces both goods, so we have joint production.
3. In addition to primary factors like labor and capital, goods enter the production process as inputs. Thus we have intermediate demand.

The full MPSGE model and MCP and NLP versions of the model are given in the Appendix. Here we reproduce and discuss only selected code snippets. The main difference to the simple model above is the specification of the production block:

$\$PROD:X(j) s:1 t:1
0:P(i) Q:make0(i,j) ! S(i,j) in the MCP and NLP models
I:P(i) Q:use0(i,j) ! D(i,j) in the MCP and NLP models
I:PF(f) Q:fd0(f,j) ! FD(f,j) in the MCP and NLP models

Note that the supply is specified in the output line, the intermediate demand is given in the first input line and the factor demand is listed in the second input line. Thus each sector $j$ produces the goods $i$ using the intermediate goods $i$ and the primary factors $f$. Consider the fully spelled-out production block for sector $s1$:

$\$PROD:X("s1") s:1 t:1
0:P("g1") Q:6.0
0:P("g2") Q:2.0
I:P("g1") Q:4.0
I:P("g2") Q:2.0
I:PF("labor") Q:1.0
I:PF("capital") Q:1.0

At benchmark, sector $s1$ produces 6 units of good $g1$ and 2 units of good $g2$. The input for this output is 4 units of the good $g1$, 2 units of the good $g2$ and labor (one unit) and capital (one unit).
Note

*Double* quotes have to be used if a singleton set element is referenced in an MPSGE model. Single quotes are not recognized.

Observe that the technology used to combine these four production factors is based on a Cobb-Douglas production function, i.e. the elasticity of substitution equals 1, as indicated by the value in field $s$. In addition, the curvature of the production possibility frontier (the constant elasticity of transformation function) is given in field $t$.

### 4.53.1.5 Modeling a Small Open Economy

An open economy is one in which goods are traded on international markets, typically at fixed international prices, i.e. there are imports and exports. One class of models for small open economies are known as "123 models" [42]. In the original 123 model, there was one small country, two producing sectors and three goods (one domestic good, one import good and one export good). The 123 model we present and discuss in this section has exogenously given world prices for imports and exports, taxes, labor-leisure choice, classical unemployment and joint production. The programming language keeps track of tax revenue flows, equations which are otherwise tedious to program. The model is a tiny version of the open economy GTAP model [115]. It follows the canonical microeconomic optimization framework: (i) consumers maximize welfare subject to a budget constraint with fixed levels of investment and public expenditure, and (ii) producers combine intermediate inputs and primary factors at least cost for given technology.

Note that the full MPSGE model mge123 and an algebraic version are given in the Appendix. In addition, we introduce and discuss a version of the model that includes nesting. Note further, that the data for the model and its variants are included in a separate GAMS file. Observe that the input-output matrix in the data file provides data on value flows for primary production factors, intermediate goods and final consumption products. In the following, we comment on selected code snippets and thereby discuss the most important features of the model.

Consider the production block for the entire production of the small economy:

```
$PROD: Y  t: etadx s: esubkl
  0: PD Q: d0 P: 1 ! YD
  0: PX Q: x0 P: px0 A: GOVT T: tx ! YX
  I: RK Q: kd0 P: rr0 A: GOVT T: tk ! KD
  I: PL Q: ly0 P: pl0 A: GOVT T: tl N: TAU_TL ! LY
```

Note that the first output line represents production for the domestic market, which is denoted by the variable $YD$ in the algebraic version of the model. The reference price at benchmark is given in the new field $P$.

Note

The default value for both P and Q fields is 1.0

The second output line represents production for the export market (variable $YX$ in the algebraic version). Observe that taxes are specified in the two new fields $A$ and $T$: the tax recipient is specified in field $A$ and the tax rate is listed in field $T$. Note that the tax recipient must reference a $\$CONSUMER$ variable and the tax rate $tx$ (tax rate on exports) is given exogenously.
Attention

In MPSGE, the way exogenous taxes are computed depends on whether they are imposed on outputs or inputs. Taxes on outputs are specified on a gross basis. Hence, if a tax on outputs has proportional rate \( t \), the producer price will be \( p(1 - t) \), where \( p \) is the market price. However, taxes on inputs are specified on a net basis. Hence, if a tax on inputs has ad valorem rate \( t \), the user cost will be \( p(1 + t) \), where \( p \) is the market price.

The first input line represents capital demand (variable \( KD \) in the algebraic version). The model includes investment by the private (domestic) households, thus the variable \( RK \) denotes a rental price index. Observe that capital is taxed exogenously at the capital tax rate \( tk \).

The second input line represents labor demand (variable \( LY \) in the algebraic version). In addition to an exogenous labor tax \( tl \), an endogenous tax is imposed on this input. The endogenous tax rate is determined by the auxiliary variable \( TAU_{TL} \), which is the entry in the field \( N \) at the end of the line. Note that an equation associated with the auxiliary variable is specified in the following block:

\[
\text{CONSTRAINT: TAU_{TL} }
\]
\[
\text{GOVT } = e = PA * g0;
\]

Note

When an auxiliary variable is fixed no constraint is required. When the lower bound of the auxiliary variable is less than the upper bound, a constraint is required.

Note further, that the variable \( TAU_{TL} \) is a wage replacement tax and at benchmark, it is fixed at zero. It is positive in two of the four counterfactual scenarios.

Domestic demand is modeled using the Armington assumption: consumers demand one single good which is an aggregation of composite goods consisting of the goods produced domestically and the imported versions, where imports and domestic goods in the same sector are imperfect substitutes. The aggregated composite good is given by a constant elasticity of substitution aggregation function of the domestic goods and the imported goods with elasticity of substitution \( \text{sigmadm} \). In our model, \( \text{sigmadm} \) equals 4 (domestic versus imported goods). The production of the Armington good is specified in the following block:

\[
\text{PROD: A s: \text{sigmadm}}
\]
\[
\text{Q:a0 A:GOVT t:ta}
\]
\[
\text{Q:d0 I:PD ! DA}
\]
\[
\text{Q:m0 p:pm0 A:GOVT t:tm ! MA}
\]

Note that both inputs are intermediate goods. The first input line represents domestic absorption (variable \( DA \) in the algebraic version of the model) and the second input line represents imports (variable \( MA \) in the algebraic version). Imports are taxed with the import tariff rate \( tm \) and the composite good is taxed with the excise and sales tax rate \( ta \). Both taxes are exogenous.

In addition, the model contains production blocks for imports and exports. Foreign exchange (denoted by the MPSGE variable \( PFX \)) is used in these two production blocks. It is the input in the production block for imports and the output in the production block for exports.

The two consumers in the model are the government (MPSGE variable \( GOVT \)) and aggregated private households (MPSGE variable \( HH \)). The demand block for the government agent is as follows:
Note that the first endowment line represents the balance of payment in the current account (denoted by the $COMMODITIES variable PFX). In our model, exports exceed imports resulting in the deficit bopdef. If imports are larger than exports there will be a surplus. The other two endowment lines represent taxes that are paid by the households: a direct tax dtax and a lumpsum labor replacement tax that is specified as an auxiliary variable and determined by the respective complementarity equation. In this model the labor replacement tax is either a lumpsum tax (TAU_LS) or it is levied on labor at a tax rate of TAU_TL (see above). Note that the direct tax and the lumpsum tax feature in the demand block for HH as negative endowments. Negative endowments represent payments to be made. The demand block for HH follows:

\$DEMAND:HH
s:sigma
E:PA Q:(-g0) R:TAU_LS
E:PA Q:(-dtax)
E:RK Q:kd0
E:PA Q:(-i0)
E:PL Q:(ly0+l0) ! Labor endowment = ly0+10 - UR * (ly0+10)
E:PL Q:(-(ly0+l0)) R:UR
D:PA Q:c0
D:PL Q:l0

Households demand two goods: the Armington composite good PA (first demand line) and leisure (second demand line). Note that the first two endowment lines represent taxes as discussed above. The third and fourth endowment line relate to capital: the third line represents the income generated by capital investment and the fourth line represents capital investments. The last two endowment lines refer to labor. Labor endowment for the aggregated households equals the sum of labor and leisure minus the percentage of unemployment. Note that at benchmark, the unemployment rate UG is fixed at zero, other scenarios are explored in the counterfactuals.

In model mge123, the rate of substitution for domestic and imported goods that constitute the Armington good PA is fixed at sigmadm = 4. Suppose that we want more control over substitution possibilities. For example, we may wish to model a substitution rate in final demand that is different than for other uses of the Armington good. MPSGE facilitates modeling this additional degree of freedom through nesting. Consider the following demand block for HH from model mgenested, a variant of model mge123:

\$DEMAND:HH
s:sigma c:sigmac
E:PA Q:(-g0) R:TAU_LS
E:PA Q:(-dtax)
E:RK Q:kd0
E:PA Q:(-i0)
E:PL Q:(ly0+l0) ! Labor endowment = ly0+10 - UR * (ly0+10)
E:PL Q:(-(ly0+l0)) R:UR
D:PD Q:cd0 c:
D:PM Q:cm0 c:
D:PL Q:l0

Note that in this formulation the endowment lines are unchanged and we have three demand lines. The first two demand lines have an additional field (c:). This tag indicates that these two goods are part of a nest: they are combined to an intermediate good called C. The rate of substitution for the goods in the nest is given in the first line of the demand block: c:sigmac. Observe that except for the substitution
rate, $C$ equals $PA$. The structure of the final demand can be presented graphically using the following nesting diagram:

\[
\begin{array}{c}
\text{U} \\
\Rightarrow \\
\ell \\
\Rightarrow \\
\sigma = \text{SIGMA} = 0.4 \\
\Rightarrow \\
C \\
\Rightarrow \\
\sigma_C = \text{SIGMAC} = 0.5 \\
\Rightarrow \\
\text{D} \\
\rightarrow \\
\text{M} \\
\rightarrow
\end{array}
\]

Figure 4.5 Structure of nested demand for HH

The nested utility function, expressed as a unit function is:

\[
U = \left[ \theta \left( \frac{\ell}{\bar{\ell}} \right)^{1-1/\sigma} + (1-\theta) \left( \alpha \left( \frac{D}{\bar{D}} \right)^{1-1/\sigma_C} + (1-\alpha) \left( \frac{M}{\bar{M}} \right)^{1-1/\sigma_C} \right)^{1-1/\sigma} \right]^{1/(1-1/\sigma)}
\]

in which $\theta$ is the budget share of leisure ($\ell$) in aggregate expenditure, and $\alpha$ is the budget share of domestic goods ($D$) in commodity expenditure, both evaluated at the benchmark point:

\[
\theta = \frac{\bar{\ell}}{\bar{\ell} + \bar{D} + \bar{M}}
\]

and

\[
\alpha = \frac{\bar{D}}{\bar{D} + \bar{M}}
\]

The demand for leisure is a function of extended income ($HH$):

\[
\ell = \bar{\ell} \left( \frac{p_U}{p_{\ell}} \right)^{\sigma} \frac{HH}{p_U(\ell + D + M)}
\]

The demand for domestic and imported goods is a function of both the top-level elasticity of substitution, $\sigma$, as well as the elasticity of substitution between consumption goods, $\sigma_C$:

\[
D = \bar{D} \left( \frac{p_U}{p_D} \right)^{\sigma} \left( \frac{p_C}{p_D} \right)^{\sigma_C} \frac{HH}{p_U(\ell + D + M)}
\]

and

\[
M = \bar{M} \left( \frac{p_U}{p_M} \right)^{\sigma} \left( \frac{p_C}{p_M} \right)^{\sigma_C} \frac{HH}{p_H(\ell + D + M)}
\]

The graph illustrates the two-level nesting structure. Domestic goods and imports are combined in the second-level nest with an elasticity of substitution of $\text{SIGMAC}$. Here we use the identifier $C$. 
Note

The name space of nesting assignments is segmented from that of sets, parameters and variables in the GAMS code. Two set identifiers are predefined: s: and t:. All other identifiers are accepted provided they have four or fewer characters.

The full model mgenested is given in the Appendix. Observe that as the substitution rate of the Armington good in final demand has changed, some parameters are modified at the beginning of the program in order to recalibrate the model.

A tariff reform is similar to a tax reform, it has a major influence on the structure of the budget of a government. Suppose we wish to keep government expenditure constant and experiment with different scenarios that represent different combinations of tax instruments. For example, at benchmark, in our model the unemployment rate UR and the endogenous wage tax TAU_TL are fixed at zero. Four alternative closures are explored in the counterfactuals. They depend on revenue replacement (lumpsum versus wage tax) and labor market (flexible versus fixed wages). An overview of the outcomes are given in the following table:

<table>
<thead>
<tr>
<th></th>
<th>Lump Sum</th>
<th>Lump Sum</th>
<th>Wage Tax</th>
<th>Wage Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flexible</td>
<td>Rigid</td>
<td>Flexible</td>
<td>Rigid</td>
</tr>
<tr>
<td>PFX</td>
<td>4.6</td>
<td>4.6</td>
<td>13.0</td>
<td>9.4</td>
</tr>
<tr>
<td>PD</td>
<td>-2.1</td>
<td>-2.1</td>
<td>5.9</td>
<td>2.6</td>
</tr>
<tr>
<td>RK</td>
<td>0.6</td>
<td>0.6</td>
<td>7.9</td>
<td>-1.6</td>
</tr>
<tr>
<td>PA</td>
<td>-4.5</td>
<td>-4.5</td>
<td>3.3</td>
<td>2.22045E-14</td>
</tr>
<tr>
<td>GOVT</td>
<td>3299.9</td>
<td>3299.9</td>
<td>3574.4</td>
<td>3458.3</td>
</tr>
<tr>
<td>HH</td>
<td>40184.6</td>
<td>40184.6</td>
<td>42403.1</td>
<td>38219.6</td>
</tr>
<tr>
<td>PX</td>
<td>4.6</td>
<td>4.6</td>
<td>13.0</td>
<td>9.4</td>
</tr>
<tr>
<td>W</td>
<td>0.4</td>
<td>0.4</td>
<td>0.3</td>
<td>-7.5</td>
</tr>
<tr>
<td>Y</td>
<td>0.3</td>
<td>0.3</td>
<td>-0.5</td>
<td>-6.3</td>
</tr>
<tr>
<td>A</td>
<td>0.7</td>
<td>0.7</td>
<td>-4.15640E-2</td>
<td>-5.3</td>
</tr>
<tr>
<td>M</td>
<td>13.7</td>
<td>13.7</td>
<td>13.0</td>
<td>7.5</td>
</tr>
<tr>
<td>X</td>
<td>18.7</td>
<td>18.7</td>
<td>17.6</td>
<td>10.2</td>
</tr>
<tr>
<td>YD</td>
<td>-8.8</td>
<td>-8.8</td>
<td>-9.5</td>
<td>-14.6</td>
</tr>
<tr>
<td>YX</td>
<td>18.7</td>
<td>18.7</td>
<td>17.6</td>
<td>10.2</td>
</tr>
<tr>
<td>KD</td>
<td>3.800191E-8</td>
<td>-7.9403E-11</td>
<td>-2.21554E-9</td>
<td>2.22045E-14</td>
</tr>
<tr>
<td>LY</td>
<td>0.6</td>
<td>0.6</td>
<td>-0.9</td>
<td>-11.9</td>
</tr>
<tr>
<td>DA</td>
<td>-8.8</td>
<td>-8.8</td>
<td>-9.5</td>
<td>-14.6</td>
</tr>
<tr>
<td>MA</td>
<td>13.7</td>
<td>13.7</td>
<td>13.0</td>
<td>7.5</td>
</tr>
<tr>
<td>C</td>
<td>1.0</td>
<td>1.0</td>
<td>-5.89421E-2</td>
<td>-7.5</td>
</tr>
<tr>
<td>LD</td>
<td>-0.9</td>
<td>-0.9</td>
<td>1.2</td>
<td>-7.5</td>
</tr>
<tr>
<td>PM</td>
<td>4.6</td>
<td>4.6</td>
<td>13.0</td>
<td>9.4</td>
</tr>
<tr>
<td>TAU_LS</td>
<td>38.1</td>
<td>38.1</td>
<td>9.1</td>
<td>11.9</td>
</tr>
<tr>
<td>TAU_TL</td>
<td></td>
<td></td>
<td>11.9</td>
<td></td>
</tr>
<tr>
<td>UR</td>
<td></td>
<td></td>
<td>10.0</td>
<td></td>
</tr>
</tbody>
</table>

Note that the classical unemployment part in this model provides a good way to motivate the usefulness of the modeling framework. We easily produce a model which illustrates the importance of the labor market formulation. When the real wage is downward rigid, replacement of tariffs with wage taxes reduces welfare (denoted by W) by 7.9%, output by 6.6% and employment by 10%. If the wage is flexible, replacement of tariffs by wage taxes leads to a 0.1% decrease in welfare.

4.53.1.6 MPSGE Keywords and Syntax

In the presentation of the general syntax in this section, we use the usual GAMS syntax symbols: [ ] (the enclosed construct is optional), { } (the enclosed construct may be repeated zero or more times) and | (exclusive OR). EOL means "end-of-line" and num_expr denotes a number or GAMS numerical expression that may include a dollar condition for exception handling.
4.53.1.6.1 MPSGE Keywords  MPSGE provides nine keywords that are used to specify an MPSGE model. They are given in Table 1. Note that these keywords are not GAMS reserved words, i.e. they are not keywords in the GAMS code apart from the MPSGE model specification.

<table>
<thead>
<tr>
<th>MPSGE Keyword and Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MODEL:model_name</td>
<td>This keyword is used to assign the identifier model_name to the model. Model_name must be a valid file name, since it is used to form model_name.GEN. This must be the first statement within the $ontext - $offtext block containing the MPSGE code; all lines in an $ontext - $offtext block before this keyword are treated as comments.</td>
</tr>
<tr>
<td>$SECTORS: sect1 ! Description1 sect2 ! Description2 ...</td>
<td>This keyword is used to declare one or more variables for sectors that are used in the model.</td>
</tr>
<tr>
<td>$COMMODITIES: com1 ! Description1 com2 ! Description2 ...</td>
<td>This keyword is used to declare one or more variables for commodities that are used in the model.</td>
</tr>
<tr>
<td>$CONSUMERS: cons1 ! Description1 cons2 ! Description2 ...</td>
<td>This keyword is used to declare one or more variables for consumers that are used in the model.</td>
</tr>
<tr>
<td>$AUXILIARY: aux1 ! Description1 aux2 ! Description2 ...</td>
<td>This keyword is used to declare one or more auxiliary variables. It is only used in models with side constraints and endogenous taxes or rationed endowments.</td>
</tr>
<tr>
<td>$PROD:sector</td>
<td>This keyword is used to specify a production function. A production function must be specified for each sector in the model. Note that sector must have been previously declared with a $SECTORS statement.</td>
</tr>
<tr>
<td>$DEMAND:consumer</td>
<td>This keyword is used to define a demand function. A demand function must be specified for each consumer in the model. Note that consumer must have been previously declared with a $CONSUMERS statement.</td>
</tr>
<tr>
<td>$CONSTRAINT:auxiliary</td>
<td>This keyword is used to specify a side constraint to be associated with the auxiliary variable auxiliary. Note that auxiliary must have been previously declared with an $AUXILIARY statement.</td>
</tr>
<tr>
<td>$REPORT:</td>
<td>This keyword identifies the set of additional variables to be calculated. These variables are used for reports and include outputs and inputs by sector, and demands and welfare by individual consumers. The variables declared in $REPORT blocks may not be used in model equations. Note that we did not discuss report blocks in this chapter. For details see [130] .</td>
</tr>
</tbody>
</table>

Table 1: MPSGE Keywords

4.53.1.6.2 Syntax for Production and Demand Functions  In addition to the keywords above, MPSGE provides a fixed structure for specifying production and demand functions. This structure has a tabular format with pre-specified fields (or labels). The names of most fields are single letters that are reserved words in MPSGE. The exception are nest identifiers: they are arbitrary names with up to four characters.

The syntax for the definition of a production function is as follows.
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1271

Each production function block starts with the MPSGE keyword $PROD$ and the respective sector. Note that the variable sector must have been previously defined in the $SECTORS$ block. The labels that follow are optional. They include the following:

- **s**: Top level elasticity of substitution between inputs. The default value is zero.
- **t**: Elasticity of transformation between outputs in production. Can be zero, but not infinity.
- **a:, b:, ...**: Elasticities of substitution in individual input nests. Here a and b are nest identifiers.

Each production block has at least one output line O and one input line I. The value in the fields O and I is a commodity that has been previously declared in the $COMMODITIES$ block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels for both lines include the following:

- **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.
- **P**: Reference price. Default value is 1.
- **A**: Tax revenue agent, the entry must be a consumer that has been previously defined in the $CONSUMERS$ block. This field must appear before the corresponding T or N field.
- **T**: Tax rate of the exogenous tax rate. Note that more than one tax rate may be listed in one line.
- **N**: Endogenous tax, the entry is an auxiliary variable that has been previously defined in the $AUXILIARY$ block.
- **M**: Endogenous tax multiplier. The ad valorem tax rate is the product of the value of the endogenous tax and this multiplier. Note that if the M field is omitted in a line with an N field, M:1 will be assumed. Observe that the M field cannot be included in the absence of an N field.
- **a:, b:, ...**: Nesting assignments. Only one such label may appear per line.

The syntax for the specification of a demand function is as follows.

Each demand function block starts with the MPSGE keyword $DEMAND$ and the respective consumer. Note that the variable consumer must have been previously declared in the $CONSUMERS$ block. The labels that follow are optional. They include the following:

- **s**: Top level elasticity of substitution between demands.
- **a:, b:, ...**: Elasticities of substitution in individual demand nests.

Each demand block has at least one demand line D and it may have one or more endowment lines E. The value in the fields D and E is a commodity that has been previously declared in the $COMMODITIES$ block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels in a D line include the following:
• **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.

• **P**: Reference price. Default value is 1.

• **a:,b:,..**: Nesting assignments. Only one such label may appear per line.

The valid labels in an **E** line include the following:

• **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.

• **R**: Rationing instrument, the entry is an auxiliary variable that has been previously defined in the **$\text{AUXILIARY}$$** block.

Auxiliary constraints in MPSGE models conform to standard GAMS equation syntax. They may refer to any of the four classes of variables, **$\text{SECTORS}$$**, **$\text{COMMODITIES}$$**, **$\text{CONSUMERS}$$** and **$\text{AUXILIARY}$$**, but they may not reference variables names declared within a **$\text{REPORT}$$** block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated **POSITIVE** (the default), the auxiliary constraint should be expressed as an equation of the type \( \leq \). If an auxiliary variable is designated **FREE**, the associated constraint must be expressed as an equality \( = \).

### 4.53.1.6.3 Domain Restrictions on Variable Declarations

Domain restrictions on variable declarations are a crucial difference between models formulated in GAMS and those formulated in MPSGE. In GAMS, variables are defined over a domain, but the explicit domain used in the model is determined by CMEX at the point when the model is generated. In an MPSGE model, users are required to declare the explicit domain for every variable. If they include variables which are not referenced in the model, they will get a "No source" or "No sink" error message. If a variable is referenced which is not declared, an error message will be generated by the MPSGE function evaluator.

Consider the following example of a sector declaration in MPSGE:

```gams
$\text{SECTORS}:
  Y(i)$y0(i)
  M(i)$m0(i)
...
```

The corresponding production blocks need to have the corresponding exception operators:

```gams
$\text{PROD}:Y(i)$y0(i)
...

$\text{PROD}:M(i)$m0(i)
...
```

For more information on exception handling in GAMS, see chapter Conditional Expressions, Assignments and Equations.
4.53.1.6.4 The Nest Labor Operator and the Spanning Operator

The nest labor operator (.tl) is used to create a set of nests. The following example serves as illustration and is self-explanatory:

```plaintext
$PROD:Y s:0.5 m:0 i.tl(m):esubdm(i) va:1
  O:PY Q:y0
  I:PD(i) Q:d0(i) i.tl:
  I:PM(i) Q:m0(i) i.tl:
  I:PK Q:k0 va:
  I:PL Q:l0 va:
```

The spanning operator (#) is a way to provide multiple inputs of one commodity. For example, a commodity with price PM is a wholesale/retail trade margin. The benchmark value of margins on commodity $i$ is md0($i$), and the net of margin (wholesale) value of commodity $i$ sales is d0($i$). If goods trade off in a Cobb-Douglas nest at the gross of margin price, we can represent this in an MPSGE model as:

```plaintext
$PROD:A s:1 i.tl:0
  O:PA Q:a0
  I:P(i) Q:d0(i) i.tl:
  I:PM#(i) Q:md0(i) i.tl:
```

4.53.1.7 MPSGE-Specific Output

The output in the listing file of an MPSGE model has some distinctive features. Note that the MPSGE model specification is reproduced in the echo print. After the directive $offtext, the echo print is interrupted and a symbol reference map is inserted. In this map, both, the identifiers defined using standard GAMS syntax and MPSGE variables are listed. In addition to the standard GAMS data types, the following MPSGE shorthand symbols may appear in this listing:

<table>
<thead>
<tr>
<th>Shorthand Symbol</th>
<th>MPSGE Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIV</td>
<td>sector variable</td>
</tr>
<tr>
<td>AUXIL</td>
<td>auxiliary variable</td>
</tr>
<tr>
<td>CONSU</td>
<td>consumer variable</td>
</tr>
<tr>
<td>PRICE</td>
<td>commodity variable</td>
</tr>
</tbody>
</table>

Table 2: Shorthand Symbols for MPSGE Data Types

For example, the symbol reference map for the first simple model in vector notation reads as follows:

Symbol Listing

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONS</td>
<td>PARAM</td>
<td>19 23</td>
</tr>
<tr>
<td>DEMAND</td>
<td>PARAM</td>
<td>20</td>
</tr>
<tr>
<td>ENDOW</td>
<td>PARAM</td>
<td>24</td>
</tr>
<tr>
<td>F</td>
<td>SET</td>
<td>9 16 16 24 24</td>
</tr>
<tr>
<td>FACTOR</td>
<td>PARAM</td>
<td>16</td>
</tr>
<tr>
<td>I</td>
<td>SET</td>
<td>3 14 15 15 16 20 20</td>
</tr>
<tr>
<td>PC</td>
<td>PRICE</td>
<td>8 15 20</td>
</tr>
<tr>
<td>PF</td>
<td>PRICE</td>
<td>9 16 24</td>
</tr>
<tr>
<td>PU</td>
<td>PRICE</td>
<td>7 19 23</td>
</tr>
</tbody>
</table>
Note that if the normalization was done automatically, a comment will appear after the solve summary detailing which variable was used as a numeraire. For example,

Default price normalization using income for RA

Observe that the lower, level, upper and marginal values of the MPSGE variables are given in the solution listing like for any other GAMS variable. For more information on standard GAMS output, see chapter GAMS Output.

4.53.1.8 Appendix

4.53.1.8.1 Three Versions of Model TWOBYTWO

4.53.1.8.1.1 MPSGE Model TWOBYTWO

$title A two by two general equilibrium model -- scalar GAMS/MPSGE

parameter  endow  index of labour endowment / 1.0 /;

* MPSGE model declaration follows

$ontext

$MODEL:twobytwo

$SECTORS:
  X ! Activity level for sector X -- benchmark=1
  Y ! Activity level for sector Y -- benchmark=1
  U ! Activity level for sector U -- benchmark=1

$COMMODITIES:
  PX ! Relative price index for commodity X -- benchmark=1
  PY ! Relative price index for commodity Y -- benchmark=1
  PU ! Relative price index for commodity U -- benchmark=1
  PL ! Relative price index for labor -- benchmark=1
  PK ! Relative price index for capital -- benchmark=1

$CONSUMERS:
  RA ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
  0:PX Q:100
  I:PL Q: 50 ! Variable LX in the algebraic model
  I:PK Q: 50 ! Variable LX in the algebraic model

$PROD:Y s:1
  0:PY Q: 50
  I:PL Q: 20 ! Variable LY in the algebraic model
  I:PK Q: 30 ! Variable KY in the algebraic model
**$PROD:U s:1**

0:PU Q:150  
I:PX Q:100 ! Variable DX in the algebraic model  
I:PY Q:50 ! Variable DY in the algebraic model

**$DEMAND:RA**

D:PU  
E:PL Q: (70*endow)  
E:PK Q: 80

**$offtext**

* Compiler directive instructing MPSGE to compile the functions

**$sysinclude mpsgeset twobytwo**

* Benchmark replication

twobytwo.iterlim = 0;  
$include TWOBYTWO.GEN  
solve twobytwo using mcp;  
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";  
twobytwo.iterlim = 1000;

* Counterfactual: 10% increase in labor endowment

endow = 1.1;

* Solve the model with the default normalization of prices which  
* fixes the income level of the representative agent. The RA  
* income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN  
solve twobytwo using mcp;

parameter equilibrium Equilibrium values;

* Save counterfactual values:

equilibrium("X.L","RA=157") = X.L;  
equilibrium("Y.L","RA=157") = Y.L;  
equilibrium("U.L","RA=157") = U.L;  
equilibrium("PX.L","RA=157") = PX.L;  
equilibrium("PY.L","RA=157") = PY.L;  
equilibrium("PU.L","RA=157") = PU.L;  
equilibrium("PL.L","RA=157") = PL.L;  
equilibrium("PK.L","RA=157") = PK.L;  
equilibrium("RA.L","RA=157") = RA.L;  
equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;  
equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;  
equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;  
equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;  
equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;  
equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;
* Fix a numeraire price index and recalculate:

\[
\begin{align*}
PX.FX &= 1; \\
\text{$\include$ TWOBYTWO.GEN} \\
\text{solve twobytwo using mcp;} \\
\text{equilibrium("X.L","PX=1") = X.L;} \\
\text{equilibrium("Y.L","PX=1") = Y.L;} \\
\text{equilibrium("U.L","PX=1") = U.L;} \\
\text{equilibrium("PX.L","PX=1") = PX.L;} \\
\text{equilibrium("PY.L","PX=1") = PY.L;} \\
\text{equilibrium("PU.L","PX=1") = PU.L;} \\
\text{equilibrium("PL.L","PX=1") = PL.L;} \\
\text{equilibrium("PK.L","PX=1") = PK.L;} \\
\text{equilibrium("RA.L","PX=1") = RA.L;} \\
\text{equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;} \\
\text{equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;} \\
\text{equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;} \\
\text{equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;} \\
\text{equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;} \\
\text{equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;} \\
\end{align*}
\]

* Recalculate with a different numeraire.
* "Unfix" the price of \( X \) and fix the wage rate:

\[
\begin{align*}
PX.UP &= +\infty; \\
PX.LO &= 1e-5; \\
PL.FX &= 1; \\
\text{$\include$ TWOBYTWO.GEN} \\
\text{solve twobytwo using mcp;} \\
\text{equilibrium("X.L","PL=1") = X.L;} \\
\text{equilibrium("Y.L","PL=1") = Y.L;} \\
\text{equilibrium("U.L","PL=1") = U.L;} \\
\text{equilibrium("PX.L","PL=1") = PX.L;} \\
\text{equilibrium("PY.L","PL=1") = PY.L;} \\
\text{equilibrium("PU.L","PL=1") = PU.L;} \\
\text{equilibrium("PL.L","PL=1") = PL.L;} \\
\text{equilibrium("PK.L","PL=1") = PK.L;} \\
\text{equilibrium("RA.L","PL=1") = RA.L;} \\
\text{equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;} \\
\text{equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;} \\
\text{equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;} \\
\text{equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;} \\
\text{equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;} \\
\text{equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;} \\
\text{display equilibrium;} \\
\end{align*}
\]

4.53.1.8.1.2 Model TWOBYTWO: Algebraic Version in GAMS MCP
$title A two by two general equilibrium model -- scalar GAMS/MCP

parameter endow index of labour endowment / 1.0 /;

* =====================================================
* Variables which appear explicitly in the MPSGE model:
* Nonnegative Variables
  X Activity level for sector X -- benchmark=1
  Y Activity level for sector Y -- benchmark=1
  U Activity level for sector U -- benchmark=1
  PU Relative price index for commodity U -- benchmark=1,
  PX Relative price index for commodity X -- benchmark=1,
  PY Relative price index for commodity Y -- benchmark=1
  PL Relative price index for labor -- benchmark=1
  PK Relative price index for capital -- benchmark=1;

Free Variable
  RA Income level for representative agent -- benchmark=150;

* Assign default prices and activity levels:
X.L = 1; Y.L = 1; U.L = 1; PX.L = 1; PY.L = 1; PK.L = 1; PU.L = 1; RA.L = 150;

* Insert lower bounds to avoid bad function calls:
PX.LO = 0.001; PY.LO = 0.001; PU.LO = 0.001; PL.LO = 0.001; PK.LO = 0.001;

* =====================================================
* Variables that enter the MPSGE model implicitly:
variables
  LX 'compensated labor demand in sector x'
  LY 'compensated labor demand in sector y'
  KX 'compensated capital demand in sector x'
  KY 'compensated capital demand in sector y'
  DX 'compensated demand for x in sector u'
  DY 'compensated demand for y in sector u';

* Equations for the implicit variables:
Equations
  lxdef 'compensated labor demand in sector x'
  lydef 'compensated labor demand in sector y'
  kxdef 'compensated capital demand in sector x'
  kydef 'compensated capital demand in sector y'
  dxdef 'compensated demand for x in sector u'
  dydef 'compensated demand for y in sector u';

lxdef..  LX =e=  50 * (PL**0.5 * PK**0.5)/PL;
lydef..  LY =e=  20 * (PL**0.4 * PK**0.6)/PL;
kxdef..  KX =e=  50 * (PL**0.5 * PK**0.5)/PK;
kydef..  KY =e=  30 * (PL**0.4 * PK**0.6)/PK;
dxdef..  DX =e= 100 * (PX**(2/3) * PY**(1/3))/PX;
dydef..    DY =e= 50 * (PX**(2/3) * PY**(1/3))/PY;

* Initial values:
LX.L = 50; LY.L = 20; KX.L = 50; KY.L = 30; DX.L = 100; DY.L = 50;

* =====================================================

Equations
prf_x  'zero profit for sector x'
prf_y  'zero profit for sector y'
prf_u  'zero profit for sector u (Hicksian welfare index)'
mkt_x  'supply-demand balance for commodity x'
mkt_y  'supply-demand balance for commodity y'
mkt_l  'supply-demand balance for primary factor l'
mkt_k  'supply-demand balance for primary factor k'
mkt_u  'supply-demand balance for aggregate demand'
i_ra   'income definition for consumer (ra)';

* Zero profit:
prf_x..    PL*LX + PK*KX =e= 100 * PX;
prf_y..    PL*LY + PK*KY =E= 50 * PY;
prf_u..    PX*DX + PY*DY =E= 150*PU;

* Market clearance:
mkt_x..    100 * X =e= DX*U;
mkt_y..    50 * Y =e= DY*U;
mkt_u..    150 * U =E= RA / PU;
mkt_l..    70 * endow =e= LX*X + LY*Y;
mkt_k..    80 =e= KX*X + KY*Y;

* Income balance:
i_ra..     RA =e= (70*endow)*PL + 80*PK;

* We declare the model using the mixed complementarity syntax
* in which equation identifiers are associated with variables.

model algebraic / prf_x.X, prf_y.Y, prf_u.U, mkt_x.PX, mkt_y.PY, mkt_l.PL,
                   mkt_k.PK, mkt_u.PU, I_ra.RA,
                   lxdef.LX, lydef.LY, kxdef.KX, kydef.KY, dxdef.DX, dydef.DY /;

* Use sector x as the numeraire commodity
PX.FX = PX.L;

algebraic.iterlim = 0;
solve algebraic using MCP;
algebraic.iterlim = 1000;

* Solve the same counterfactual:
endow = 1.1;
* Fix the income level at the default level, i.e. the income level corresponding to the counterfactual endowment at benchmark price:

\[ RA.FX = 80 + 1.1 \times 70; \]

solve algebraic using MCP;

parameter equilibrium Equilibrium values;

* Save counterfactual values:

\[ \text{equilibrium("X.L","RA=157") = X.L;} \]
\[ \text{equilibrium("Y.L","RA=157") = Y.L;} \]
\[ \text{equilibrium("U.L","RA=157") = U.L;} \]
\[ \text{equilibrium("PX.L","RA=157") = PX.L;} \]
\[ \text{equilibrium("PY.L","RA=157") = PY.L;} \]
\[ \text{equilibrium("PU.L","RA=157") = PU.L;} \]
\[ \text{equilibrium("PL.L","RA=157") = PL.L;} \]
\[ \text{equilibrium("PK.L","RA=157") = PK.L;} \]
\[ \text{equilibrium("RA.L","RA=157") = RA.L;} \]
\[ \text{equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;} \]
\[ \text{equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;} \]
\[ \text{equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;} \]
\[ \text{equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;} \]
\[ \text{equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;} \]
\[ \text{equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;} \]

* Fix a numeraire price index and recalculate:

\[ RA.LO = -\infty; \]
\[ RA.UP = \infty; \]
\[ PX.FX = 1; \]

solve algebraic using mcp;

\[ \text{equilibrium("X.L","PX=1") = X.L;} \]
\[ \text{equilibrium("Y.L","PX=1") = Y.L;} \]
\[ \text{equilibrium("U.L","PX=1") = U.L;} \]
\[ \text{equilibrium("PX.L","PX=1") = PX.L;} \]
\[ \text{equilibrium("PY.L","PX=1") = PY.L;} \]
\[ \text{equilibrium("PU.L","PX=1") = PU.L;} \]
\[ \text{equilibrium("PL.L","PX=1") = PL.L;} \]
\[ \text{equilibrium("PK.L","PX=1") = PK.L;} \]
\[ \text{equilibrium("RA.L","PX=1") = RA.L;} \]
\[ \text{equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;} \]
\[ \text{equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;} \]
\[ \text{equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;} \]
\[ \text{equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;} \]
\[ \text{equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;} \]
\[ \text{equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;} \]
* Recalculate with a different numeraire.
* "Unfix" the price of X and fix the wage rate:

PX.UP = +inf;
PX.L0 = 1e-5;
PL.FX = 1;
solve algebraic using mcp;
equilibrium("X.L","PL=1") = X.L;
equilibrium("Y.L","PL=1") = Y.L;
equilibrium("U.L","PL=1") = U.L;
equilibrium("PX.L","PL=1") = PX.L;
equilibrium("PY.L","PL=1") = PY.L;
equilibrium("PU.L","PL=1") = PU.L;
equilibrium("PL.L","PL=1") = PL.L;
equilibrium("PK.L","PL=1") = PK.L;
equilibrium("RA.L","PL=1") = RA.L;
equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;
equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;
display equilibrium;

4.53.1.8.1.3 Indexed MPSGE Model TWOBYTWO

$title: A two by two general equilibrium model -- indexed GAMS/MPSGE

Sets
i Produced goods / x, y /,
f Factors of production / L, K /;
table sam(*,*) Benchmark input-output matrix
   X  Y  U  RA
X  100 -100
Y  50  -50
U  150 -150
K -50  -20  70
L -50  -30  80;

parameters
supply(i) Benchmark supply of output of sectors,
factor(f,i) Benchmark factor demand,
demand(i) Benchmark demand for consumption,
endow(f) Factor endowment,
cons Benchmark total consumption;
* Extract data from the original format into model-specific arrays

supply(i) = sam(i,i);
factor(f,i) = -sam(f,i);
demand(i) = -sam(i,'u');
cons = sum(i, demand(i));
endow(f) = sam(f,'ra');

display supply, factor, demand, cons, endow;

$ontext
$MODEL:twobytwo

$SECTORS:
   Y(i) ! Activity level -- benchmark=1
   U  ! Final consumption index -- benchmark=1

$COMMODITIES:
   PU  ! Relative price of final consumption -- benchmark=1
   PC(i) ! Relative price of commodities -- benchmark=1
   PF(f) ! Relative price of factors -- benchmark=1

$CONSUMERS:
   RA ! Income level (benchmark=150)

$PROD:Y(i) s:1
   O:PC(i) Q:supply(i)
   I:PF(f) Q:factor(f,i)

$PROD:U s:1
   O:PU Q:cons
   I:PC(i) Q:demand(i)

$DEMAND:RA
   D:PU Q:cons
   E:PF(f) Q:endow(f)

$offtext
$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

endow('l') = 1.1*endow('l');

* Solve the model with the default normalization of prices which
* fixes the income level of the representative agent. The RA
* income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

parameter equilibrium Equilibrium values;

* Save counterfactual values:
equilibrium("Y.L",i,"RA=157") = Y.L(i);
equilibrium("U.L","_","RA=157") = U.L;
equilibrium("PC.L",i,"RA=157") = PC.L(i);
equilibrium("PF.L",f,"RA=157") = PF.L(f);
equilibrium("RA.L","_","RA=157") = RA.L;
equilibrium("PC(i)/PC("x")",i,"RA=157") = PC.L(i)/PC.L("x");
equilibrium("PF(f)/PC("x")",f,"RA=157") = PF.L(f)/PC.L("x");
equilibrium("RA.L/PC("x")","_","RA=157") = RA.L/PC.L("x");

* Fix a numeraire price index and recalculate:

PC.FX("x") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

* Recalculate with a different numeraire.
* "Unfix" the price of X and fix the wage rate:

PC.UP("X") = +inf; PC.LO("X") = 1e-5; PF.FX("L") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

4.53.1.8.2 Three Versions of Model JPMGE

4.53.1.8.2.1 MPSGE Model JPMGE

$title Model with Joint Products and Intermediate Demand -- solved with GAMS/MPSGE

Sets j Sectors / s1*s2 /,
i Goods / g1*g2 /,
f Primary factors / labor, capital / ;
alias (i,ii),(j,jj);
Table make0(i,j) Matrix -- supplies
   s1  s2  
g1  6   2  
g2  2   10;

Table use0(i,j) Use matrix -- intermediate demands
   s1  s2  
g1  4   2  
g2  2   6;

Table fd0(f,j) Factor demands
   s1  s2  
labor 1   3  
capital 1  1;

Parameters
   c0(i) Consumer demand / g1 2, g2 4 /
   e0(f) Factor endowments;

\( e0(f) = \sum(j, fd0(f,j)) \)

display e0;

$ontext
$MODEL:jpmge

$SECTORS:
   X(j) ! Activity index -- benchmark=1

$COMMODITIES:
   P(i) ! Relative commodity price -- benchmark=1
   PF(f) ! Relative factor price -- benchmark=1

$CONSUMERS:
   Y ! Nominal household income=expenditure

$PROD:X(j) s:1 t:1
   O:P(i) Q:make0(i,j) ! S(i,j) in the MCP and NLP models
   I:P(i) Q:use0(i,j) ! D(i,j) in the MCP and NLP models
   I:PF(f) Q:fd0(f,j) ! FD(f,j) in the MCP and NLP models

$REPORT:
   v:S(i,j) O:P(i) PROD:X(j)
   v:D(i,j) I:P(i) PROD:X(j)
   v:FD(f,j) I:PF(f) PROD:X(j)

$DEMAND:Y s:1
   D:P(i) Q:c0(i)
   E:PF(f) Q:e0(f)

$REPORT:
   v:C(i) D:P(i) DEMAND:Y

$offtext
$sysinclude mpsgeset jpmge

* Benchmark replication
jmpg.iterlim = 0;
$include JPMGE.GEN
solve jmpge using mcp;
abort$(abs(jpmge.objval) gt 1e-7) "JPMGE does not calibrate!";
jpmge.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

$include JPMGE.GEN
solve jmpge using mcp;

Parameter equilibrium Equilibrium values;

* Save counterfactual values:

   equilibrium("X",j,"Y=6.4") = X.L(j);
equilibrium(i,j,"Y=6.4") = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4") = FD.L(f,j);
equilibrium("C",i,"Y=6.4") = C.L(i);
equilibrium("X",j,"Y=6.4") = X.L(j);
equilibrium("P",i,"Y=6.4") = P.L(i);
equilibrium("PF",f,"Y=6.4") = PF.L(f);
equilibrium("Y","_","Y=6.4") = Y.L;

* Fix a numeraire price index and recalculate:

P.FX("g1") = 1;
$include JPMGE.GEN
solve jmpge using mcp;

   equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium(i, j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'P("g1")=1') = FD.L(f,j);
equilibrium("C", i,'P("g1")=1') = C.L(i);
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium("P", i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_','P("g1")=1') = Y.L;

* Recalculate with a different numeraire.
* "Unfix" the price of X and fix the wage rate:

P.UP("g1") = +inf;
P.LO("g1") = 1e-5;
P.FX("labor") = 1;

$include JPMGE.GEN
solve jmpge using mcp;

   equilibrium("X", j,'PF("labor")=1') = X.L(j);
equilibrium(i, j,'PF("labor")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'PF("labor")=1') = FD.L(f,j);
equilibrium("C", i,'PF("labor")=1') = C.L(i);
equilibrium("X", j,'PF("labor")=1') = X.L(j);
equilibrium("P", i,'PF("labor")=1') = P.L(i);
equilibrium("PF", f,'PF("labor")=1') = PF.L(f);
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE)

equilibrium("Y","_","PF("labor")=1") = Y.L;

option equilibrium:3:2:1;
display equilibrium;

4.53.1.8.2.2 Algebraic Version of Model JPMGE: MCP Formulation

$title Model with Joint Products and Intermediate Demand -- solved with GAMS/MCP

Sets
  j     Sectors / s1*s2 /,
  i     Goods / g1*g2 /,
  f     Primary factors / labor, capital /;
alias (i,ii),(j,jj);

Table make0(i,j) Matrix -- supplies
  s1   s2
  g1   6   2
  g2   2   10;

Table use0(i,j) Use matrix -- intermediate demands
  s1   s2
  g1   4   2
  g2   2   6;

Table fd0(f,j) Factor demands
  s1   s2
  labor 1   3
  capital 1  1;

Parameters
  c0(i) Consumer demand / g1 2, g2 4 /
  e0(f) Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;

* ==============================================================
* Variables which appear explicitly in the MPSGE model:

Variables
  X(j) ! Activity index -- benchmark=1
  P(i) ! Relative commodity price -- benchmark=1
  PF(f) ! Relative factor price -- benchmark=1
  Y    ! Nominal household income=expenditure;

X.L(j) = 1; P.L(i) = 1; PF.L(f) = 1; Y.L = sum(f,e0(f));
P.L0(i) = 1e-4; PF.L0(f) = 1e-4;

* ==============================================================
* Variables that enter the MPSGE model implicitly:

Variables
  S(i,j) Compensated supply
  D(i,j) Compensated intermediate demand
  FD(f,j) Compensated factor demand
  C(i)   Final demand;
S.L(i,j) = make0(i,j);
D.L(i,j) = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i) = c0(i);

* Calibration calculations provided automatically by MPSGE:

Parameter thetad(i,j) Intermediate demand value share
  thetas(i,j) Output value share
  thetaf(f,j) Factor demand value share
  thetac(i) Final demand value share;

thetas(i,j) = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j) = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j) = fd0(f,j) /(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));

thetac(i) = c0(i)/sum(ii,c0(ii));

alias (i,i_), (f,f_);

* Equations for the implicit variables:

Equations sdef, ddef, fddef, cdef;

$macro REV(j) (sqrt(sum(i_,thetas(i_,j)*sqr(P(i_)))))

sdef(i,j) .. S(i,j) =e= make0(i,j)*P(i)/REV(j);

$macro COST(j) (prod(i_,P(i_))**thetad(i_,j)*prod(f_,PF(f_)**thetaf(f_,j)))

ddef(i,j) .. D(i,j) =e= use0(i,j) * COST(j)/P(i);

fddef(f,j) .. FD(f,j) =e= fd0(f,j) * COST(j)/PF(f);

cdef(i) .. C(i) =e= thetac(i) * Y/P(i);

* Equilibrium conditions:

Equations prf_X(j), mkt_p(i), mkt_pf(f), income;

* Zero profit:

prf_X(j) .. sum(f,PF(f)*FD(f,j)) + sum(i,P(i)*D(i,j)) =e= sum(i,P(i)*S(i,j));

* Market clearance:

mkt_P(i) .. sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);

mkt_PF(f) .. e0(f) =e= sum(j, FD(f,j)*X(j));

* Income balance:

income .. Y =e= sum(f, PF(f)*e0(f));

* Benchmark replication with iteration limit zero. Do not need to fix the price level at this point:

```plaintext
jpmcp.iterlim = 0;
solve jpmcp using mcp;
abort$(abs(jpmcp.objval) gt 1e-7) "JPMCP does not calibrate!";
jpmcp.iterlim = 1000;
```

* Counterfactual: 10% increase in labor endowment

```plaintext
e0("labor") = 1.1 * e0("labor");
```

* Fix the income level at the default level, i.e. the income level corresponding to the counterfactual endowment at benchmark price:

```plaintext
Y.FX = sum(f,e0(f));
solve jpmcp using mcp;
```

Parameter equilibrium Equilibrium values;

* Save counterfactual values:

```plaintext
equilibrium("X",j,"Y=6.4") = X.L(j);
equilibrium(i,j,"Y=6.4") = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4") = FD.L(f,j);
equilibrium("C",i,"Y=6.4") = C.L(i);
equilibrium("X",j,"Y=6.4") = X.L(j);
equilibrium("P",i,"Y=6.4") = P.L(i);
equilibrium("PF",f,"Y=6.4") = PF.L(f);
equilibrium("Y","_","Y=6.4") = Y.L;
```

* Fix a numeraire price index and recalculate:

```plaintext
P.FX("g1") = 1;
Y.L0 = -INF;
Y.UP = INF;
solve jpmcp using mcp;
```

```plaintext
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium(i, j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'P("g1")=1') = FD.L(f,j);
equilibrium("C", i,'P("g1")=1') = C.L(i);
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium("P", i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_","P("g1")=1") = Y.L;
```

* Recalculate with a different numeraire.

* "Unfix" the price of X and fix the wage rate:

```plaintext
P.UP("g1") = +inf;
```
P.LO("g1") = 1e-5;
P.F.FX("labor") = 1;

solve jpmcp using mcp;

equilibrium("X", j, 'PF("labor")=1') = X.L(j);
equilibrium(i, j, 'PF("labor")=1') = S.L(i,j) - D.L(i,j);
equilibrium(f, j, 'PF("labor")=1') = FD.L(f,j);
equilibrium("C", i, 'PF("labor")=1') = C.L(i);
equilibrium("P", i, 'PF("labor")=1') = P.L(i);
equilibrium("PF", f, 'PF("labor")=1') = PF.L(f);
equilibrium("Y", _, 'PF("labor")=1') = Y.L;

option equilibrium:3:2:1;
display equilibrium;

4.53.1.8.2.3 Algebraic Version of Model JPMGE: NLP Formulation

$title Model with Joint Products and Intermediate Demand -- solved as NLP

Sets
  j  Sectors  / s1*s2 /,
i  Goods  / g1*g2 /,
f  Primary factors  / labor, capital / ;
alias (i,ii),(j,jj);

Table make0(i,j) Matrix -- supplies
  s1  s2
  g1  6  2
  g2  2 10 ;

Table use0(i,j) Use matrix -- intermediate demands
  s1  s2
  g1  4  2
  g2  2  6 ;

Table fd0(f,j) Factor demands
  s1  s2
  labor  1  3
  capital 1  1 ;

Parameters
  c0(i) Consumer demand  / g1 2, g2 4 /
e0(f) Factor endowments;

  e0(f) = sum(j, fd0(f,j));
display e0;

* ==============================================================
* Variables which appear explicitly in the MPSGE model:

Nonnegative
Variable  X(j) ! Activity index -- benchmark=1;
  X.L(j) = 1;
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE)

* Variables that enter the MPSGE model implicitly:

Variables
- U: Utility
- S(i,j): Compensated supply
- D(i,j): Compensated intermediate demand
- FD(f,j): Compensated factor demand
- C(i): Final demand;

S.L(i,j) = make0(i,j);
D.L(i,j) = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i) = c0(i);

* Calibration calculations provided automatically by MPSGE:

Parameter
- thetad(i,j): Intermediate demand value share
- thetas(i,j): Output value share
- thetaf(f,j): Factor demand value share
- thetac(i): Final demand value share;

thetas(i,j) = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j) = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j) = fd0(f,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetac(i) = c0(i)/sum(ii,c0(ii));

* Equilibrium conditions:

* Zero profit:

production(j).. prod(i, (D(i,j)/use0(i,j))**thetad(i,j)) * prod(f, (FD(f,j)/fd0(f,j))**thetaf(f,j)) =e= sqrt(sum(i, thetas(i,j)*sqr(S(i,j)/make0(i,j))));

* Market clearance:

goods(i).. sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);
factors(f).. e0(f) =e= sum(j, FD(f,j)*X(j));

* Income balance:

utility.. U =E= prod(i, (C(i)/c0(i))**thetac(i));

model jpnlp / production, goods, factors, utility/;

* Benchmark replication with iteration limit zero. Do not need
* to fix the price level at this point:

solve jpnlp using nlp maximizing U;

Parameter equilibrium Equilibrium values
pnum

* Save benchmark values:

\[
pnum = \frac{\sum(f, \text{factors.m}(f) \cdot e0(f))}{\sum(f, e0(f))};
\]

\[
equilibrium("X",j,"bmk") = X.L(j);
\]

\[
equilibrium(i,j,"bmk") = X.L(j) \cdot (S.L(i,j) - D.L(i,j));
\]

\[
equilibrium(f,j,"bmk") = FD.L(f,j);
\]

\[
equilibrium("C",i,"bmk") = C.L(i);
\]

\[
equilibrium("X",j,"bmk") = X.L(j);
\]

\[
equilibrium("P",i,"bmk") = \text{goodsm}(i)/pnum;
\]

\[
equilibrium("PF",f,"bmk") = \text{factors.m}(f)/pnum;
\]

\[
equilibrium("Y",_,"bmk") = \sum(f, \text{factors.m}(f) \cdot e0(f)/pnum);
\]

* Counterfactual : 10% increase in labor endowment

\[
e0("labor") = 1.1 \times e0("labor");
\]

solve jpmpl using nlp maximizing u;

* Save counterfactual values:

\[
pnum = \frac{\sum(f, \text{factors.m}(f) \cdot e0(f))}{\sum(f, e0(f))};
\]

\[
equilibrium("X",j,"labor+10\%") = X.L(j);
\]

\[
equilibrium(i,j,"labor+10\%") = X.L(j) \cdot (S.L(i,j) - D.L(i,j));
\]

\[
equilibrium(f,j,"labor+10\%") = FD.L(f,j);
\]

\[
equilibrium("C",i,"labor+10\%") = C.L(i);
\]

\[
equilibrium("X",j,"labor+10\%") = X.L(j);
\]

\[
equilibrium("P",i,"labor+10\%") = \text{goodsm}(i)/pnum;
\]

\[
equilibrium("PF",f,"labor+10\%") = \text{factors.m}(f)/pnum;
\]

\[
equilibrium("Y",_,"labor+10\%") = \sum(f, \text{factors.m}(f) \cdot e0(f)/pnum);
\]

option equilibrium:3:2:1;

display equilibrium;

4.53.1.8.3 Three Versions of a 123 Model

4.53.1.8.3.1 Data for Models: 123DATA This data file is included in each of the following models.

$stitle Dataset for a 123 Model

set mcmrow Rows in the micro-consistent matrix /
  PFX Current account,
  PD Domestic output
  TA Sales and excise taxes
  TM Import tariffs
  TX Export taxes
  TK Capital taxes
  TL Labor taxes
  RK Return to capital
  PL Wage rate
  PA Price of Armington composite /,
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1291

mcmcol Columns in the micro-consistent matrix /
  S Supply,
  D Demand,
  GOVT Government,
  HH Households
  INVEST Investment /;

table mcm(mcmrow,mcmcol) Microconsistent matrix

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>D</th>
<th>GOVT</th>
<th>HH</th>
<th>INVEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFX</td>
<td>106.386</td>
<td>-144.701</td>
<td>38.315</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td>218.308</td>
<td>-218.308</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>-32.027</td>
<td>32.027</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td>-18.617</td>
<td>18.617</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TX</td>
<td>-1.136</td>
<td>1.136</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TK</td>
<td>-12.837</td>
<td>12.837</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TL</td>
<td>-3.539</td>
<td>3.539</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RK</td>
<td>-143.862</td>
<td>143.862</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td>-163.320</td>
<td>163.320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>413.653</td>
<td>-35.583</td>
<td>-291.694</td>
<td>-86.376</td>
<td></td>
</tr>
</tbody>
</table>

* Parameter values describing base year equilibrium:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>px0</td>
<td>Reference price of exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d0</td>
<td>Reference domestic supply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>Reference exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kd0</td>
<td>Reference net capital earnings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ly0</td>
<td>Reference net labor earnings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rr0</td>
<td>Reference price of capital</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p10</td>
<td>Reference wage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tk</td>
<td>Capital tax rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tl</td>
<td>Labor tax rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta</td>
<td>Excise and sales tax rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx</td>
<td>Tax on exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a0</td>
<td>Aggregate supply (gross of tax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g0</td>
<td>Government demand,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtax</td>
<td>Direct tax net transfers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m0</td>
<td>Imports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l0</td>
<td>Leisure demand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c0</td>
<td>Household consumption,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i0</td>
<td>Aggregate investment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tm</td>
<td>Import tariff rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pm0</td>
<td>Reference price of imports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pwm</td>
<td>World price of imports /1/</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pwx</td>
<td>World price of exports /1/</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bopdef</td>
<td>Balance of payments deficit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>etadx</td>
<td>Elasticity of transformation (D versus X) /4/,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigmadm</td>
<td>Elasticity of substitution (D versus M) /4/,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>esubkl</td>
<td>Elasticity of substitution (K versus L) /1/,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>Elasticity of substitution (C versus LS) /0.4/;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

d0 = mcm("pd","s");
x0 = mcm("pfx","s");
kd0 = -mcm("rk","s");
ly0 = -mcm("pl","s");

tx = -mcm("tx","s")==mcm("pfx","s");
tk = mcm("tk", "s")/mcm("rk", "s");
tl = mcm("tl", "s")/mcm("pl", "s");

px0 = 1 - tx;
rr0 = 1 + tk;
pl0 = 1 + tl;

parameter profit Zero profit check;
profit("PD") = d0;
profit("PX") = x0;
profit("TX") = -tx*x0;
profit("TK") = -tk*kd0;
profit("TL") = -tl*ly0;
profit("PL") = -ly0;
profit("RK") = -kd0;

alias (u,*);
profit("CHK") = sum(u, profit(u));
display profit, tx, tk, tl;

m0 = -mcm("pfx","d");
tm = mcm("tm","d")/mcm("pfx","d");
pm0 = 1 + tm;
a0 = mcm("pa","d");
g0 = -mcm("pa","govt");
ta = -mcm("ta","d")/mcm("pa","d");
bopdef = mcm("pfx","govt");
dtax = g0 - bopdef - tm*m0 - ta*a0 - tl*ly0 - tk*kd0 - tx*x0;
i0 = -mcm("pa","invest");
c0 = a0 - i0 - g0;
l0 = 0.75*ly0;
display g0;

4.53.1.8.3.2 MGE123: MPSGE Model  This is the base 123 MPSGE model.

$title Static 123 Model Ala Devarjan
$include 123data.gms
$ontext
$model:MGE123
$SECTORS:  
Y  ! Production
A  ! Armington composite
M  ! Imports
X  ! Exports
$COMMODITIES:  
PD  ! Domestic price index
PX  ! Export price index
PM  ! Import price index
PA  ! Armington price index
PL  ! Wage rate index
RK  ! Rental price index
PFX ! Foreign exchange
$CONSUMERS:
    HH        ! Private households
    GOVT     ! Government

$AUXILIARY:
    TAU_LS    ! Lumpsum Replacement tax
    TAU_TL    ! Labor tax replacement
    UR        ! Unemployment rate

$PROD:Y t:etadx s:esubkl
    0:PD Q:d0 P:1        ! YD
    0:PX Q:x0 P:px0 A:GOVT T:tx    ! YX
    I:RK Q:kd0 P:rr0 A:GOVT T:tk    ! KD
    I:PL Q:ly0 P:pl0 A:GOVT T:tl N:TAU_TL   ! LY

$report:
    v:YD o:PD prod:Y
    v:YX o:PX prod:Y
    v:KD i:RK prod:Y
    v:LY i:PL prod:Y

$PROD:A s:sgmadm
    0:PA Q:a0 A:GOVT t:ta
    I:PD Q:d0        ! DA
    I:PM Q:m0 p:pm0 A:GOVT t:tm   ! MA

$report:
    v:DA i:PD prod:A
    v:MA i:PM prod:A

$PROD:M
    0:PM Q:m0
    I:PFX Q:(pwm*m0)

$PROD:X
    0:PFX Q:(pwx*x0)
    I:PX Q:x0

$DEMAND:GOVT
    E:PFX Q:bopdef
    E:PA Q:dtax
    E:PA Q:g0     R:TAU_LS
    D:PA

$CONSTRAINT:UR
    PL =q= PA;

$CONSTRAINT:TAU_LS
    GOVT =e= PA * g0;

$CONSTRAINT:TAU_TL
    GOVT =e= PA * g0;

$DEMAND:HH s:sgma
    E:PA Q:(-g0)     R:TAU_LS
    E:PA Q:(-dtax)
    E:RK Q:kd0
    E:PA Q:(-i0)
E:PL Q: (ly0+10) ! Labor endowment = ly0+10 - UR * (ly0+10)
E:PL Q: (-ly0+10) R:UR
D:PA Q: c0
D:PL Q: 10

$report:
v:W w:HH
v:C d:PA demand:HH
v:LD d:PL demand:HH

$offtext
$sysinclude mpsgeset mge123

UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP = INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate."

mge123.iterlim = 10000;

Parameter report Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve."

report("W","%replacement%","%labormarket%") = 100*(W.L-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("M","%replacement%","%labormarket%") = 100*(M.L-1);
report("X","%replacement%","%labormarket%") = 100*(X.L-1);
report("YD","%replacement%","%labormarket%") = 100*(YD.L/d0-1);
report("YY","%replacement%","%labormarket%") = 100*(YY.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100*(KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100*(LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100*(DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100*(MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100*(C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100*(LD.L/10-1);
report("PD","%replacement%","%labormarket%") = 100*(PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100*(PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100*(PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100*(PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100*(PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100*(RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100*(PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100*(HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100*(GOVT.L/PL.L - 1);
report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;

$offecho
Tariff reform:

\[ \text{tm} = 0; \]
\[ \text{UR.FX} = 0; \]
\[ \text{TAU_LS.UP} = +\infty; \]
\[ \text{TAU_LS.LO} = -\infty; \]
\[ \text{TAU_TL.FX} = 0; \]
\[ \text{\$include MGE123.GEN} \]
\[ \text{solve mge123 using mcp; } \]

\$set replacement Lump Sum
\$set labormarket Flexible
\$include \%\text{\texttt{gams.scrdir}}\%\text{\texttt{report}}

\[ \text{UR.FX} = 0; \]
\[ \text{TAU_TL.UP} = +\infty; \]
\[ \text{TAU_TL.LO} = -\infty; \]
\[ \text{TAU_LS.FX} = 0; \]
\[ \text{\$include MGE123.GEN} \]
\[ \text{solve mge123 using mcp; } \]

\$set replacement Wage Tax
\$set labormarket Flexible
\$include \%\text{\texttt{gams.scrdir}}\%\text{\texttt{report}}

\[ \text{UR.LO} = 0; \]
\[ \text{UR.UP} = +\infty; \]
\[ \text{TAU_LS.UP} = +\infty; \]
\[ \text{TAU_LS.LO} = -\infty; \]
\[ \text{TAU_TL.FX} = 0; \]
\[ \text{\$include MGE123.GEN} \]
\[ \text{solve mge123 using mcp; } \]

\$set replacement Lump Sum
\$set labormarket Rigid Wage
\$include \%\text{\texttt{gams.scrdir}}\%\text{\texttt{report}}

\[ \text{UR.LO} = 0; \]
\[ \text{UR.UP} = +\infty; \]
\[ \text{TAU_TL.UP} = +\infty; \]
\[ \text{TAU_TL.LO} = -\infty; \]
\[ \text{TAU_LS.FX} = 0; \]
\[ \text{\$include MGE123.GEN} \]
\[ \text{solve mge123 using mcp; } \]

\$set replacement Wage Tax
\$set labormarket Rigid Wage
\$include \%\text{\texttt{gams.scrdir}}\%\text{\texttt{report}}

\text{option report:1:1:2;}
\text{display report;}

4.53.1.8.3.3 Model MCP123: Algebraic Version in GAMS MCP

This GAMS model is the model mge123 translated verbatim into GAMS/MCP (algebraic) format. This is the way the model will look in the MPSGEv2 framework.

\text{\$title Static 123 Model Ala Devarjan -- GAMS/MCP Format}
$include 123data

* Variables which appear explicitly in the MPSGE model:

Nonnegative Variables

*$SECTORS:
  Y Production
  A Armington composite
  M Imports
  X Exports

*$COMMODITIES:
  PD Domestic price index
  PX Export price index
  PM Import price index
  PA Armington price index
  PL Wage rate index
  RK Rental price index
  PFX Foreign exchange

*$CONSUMERS:
  HH Private households
  GOVT Government

*$AUXILIARY:
  TAU_TL Wage replacement tax,
  TAU_LS Lump sum replacement tax
  UR Unemployment rate;

* Assign default prices and activity levels:

  Y.L = 1; A.L = 1; M.L = 1; X.L = 1;
  PD.L = 1; PX.L = 1; PM.L = 1; PA.L = 1; PL.L = 1; RK.L = 1; PFX.L = 1;
  HH.L = c0+l0; GOVT.L=g0;
  TAU_TL.L = 0; TAU_LS.L = 0; UR.L = 0;

* Insert lower bounds to avoid bad function calls:

  PD.LO = 1e-4; PX.LO = 1e-4; PM.LO = 1e-4; PA.LO = 1e-4; PL.LO = 1e-4; RK.LO = 1e-4; PFX.LO = 1e-4;

* Variables enter the MPSGE model implicitly:

Variable
  YD Production for the domestic market,
  YX Production for the export market,
  KD Capital demand,
  LY Labor demand,
  DA Domestic absorption,
  MA Imports,
  C Consumption of goods (uncompensated),
  LD Leisure demand (uncompensated);

* Equations for the implicit variables:
Equations YDdef, YXdef, KDdef, LYdef, DAdef, MAdef, Cdef, LDdef;

* Macros defining composite prices (unit cost and unit revenue):

Parameter thetal Labor share in cost function, 
thetac Consumption share in expenditure function, 
thetam Share parameter in Armington function 
thetaz Share parameter in transformation function ; 

thetal = ly0*pl0 /(kd0*rr0+ly0*pl0); 
thetaz = x0 *px0 /((d0+x0*px0); 
thetam = m0 *pm0 /((d0+m0*pm0); 
thetac = c0/(c0+10);

$macro CY ((PL*(1+tl+TAU_TL)/pl0)**thetal * (RK*(1+tk)/rr0)**(1-thetal))$macro RY ((thetaz * (PX*(1-tx)/px0)**(1+etadx) + (1-thetaz) * PD**(1+etadx))**(1/(1+etadx)))$macro CA ((thetam *(PM*(1+tm)/pm0)**(1-sigmadm) + (1-thetam)*PD**(1-sigmadm))**(1/(1-sigmadm)))$macro CU ((thetac*PA**(1-sigma) + (1-thetac)*PL**(1-sigma))**(1/(1-sigma)))$macro W (HH.L/((c0+l0)*CU))

* Definitions of demand and supply functions:

YDdef.. YD =e= d0 * (PD/RY)**etadx; 
YXdef.. YX =e= x0 * (PX*(1-tx)/px0*RY)**etadx; 
KDdef.. KD =e= kd0 * (CY*rr0/(RK*(1+tk)))**esubkl; 
LYdef.. LY =e= ly0 * (CY*pl0/(PL*(1+tl+TAU_TL)))**esubkl; 

DAdef.. DA =e= d0 * (CA/PD)**sigmadm; 
MAdef.. MA =e= m0 * (CA*pm0/(PM*(1+tm)))**sigmadm; 

Cdef.. C =e= c0 * W * (CU/PA)**sigma; 
LDdef.. LD =e= 10 * W * (CU/PL)**sigma; 

* Initialize: 

YD.L = d0; YX.L = x0; KD.L = kd0; LY.L = ly0; DA.L = d0; MA.L = m0; 
C.L = c0; LD.L = 10; 

* ============================================================== 

Equations 

* Zero profit condition 

profitd domestic production, 
profita Armington supply, 
profitm imported goods production 
profitx exported goods production 

* Market clearing condition 

marktd domestic goods market, 
marketa Armington goods market 
marktm imported goods market 
marktx exported goods market 
marktfx balance of payment 
marketk capital market 
marketl labor market
* Income balance
  incomeg budget
  incomeh household budget

* Additional constraints
  tauTLdef Equal yield constraint (TL),
  tauLSdef Equal yield constraint (LS),
  URdef Lower bound on the real wage;

marketd.. Y*YD =e= A*DA;
profity.. KD*RK*(1+tk) + LY*PL*(1+tl+TAU_TL) =E= YD*PD + YX*PX*(1-tx);
marketa.. A*a0 =g= GOVT/PA + i0 + C;
profita.. PD*DA + PM*(1+tm)*MA =e= PA*a0*(1-ta);
marketa.. M*m0 =e= A*MA;
profitm.. PFX*pwm =e= PM;
marketx.. Y*YX =e= X*x0;
profitx.. PX =e= PFX*pwx;
marketfx.. X*pwx*x0 + bopdef =E= M*pwm*m0;
marketk.. kd0 =e= Y*KD;
marketl.. ly0+10 =e= Y*LY + LD + (ly0+10)*UR;
incomeg.. GOVT =e= PFX*bopdef + PA*dtax + PA*g0*TAU_LS + tx*PX*YX*Y +
  tk*RK*KD*Y + (tl+TAU_TL)*PL*LY*Y + tm*PM*MA*A + ta*PA*a0*A;
tauTLdef.. GOVT =e= PA * g0;
tauLSdef.. GOVT =e= PA * g0;
URdef.. PL =G= PA;
incomeh.. HH =e= PL*(ly0+10)*(1-UR) - PA*dtax - PA*g0*TAU_LS + RK*kd0 - PA*i0 ;

model mcp123 /marketd.PD, marketa.PA, marketm.PM, marketx.PX, marketfx.PFX, marketk.RK, marketl.PL,
  YDdef.YD, YXdef.YX, KDdef.KD, LYdef.LY, DAdef.DA, MAdf.MA, Cdef.C, LDdef.LD /;

* Establish a numeraire price index:

HH.FX = HH.L;
mcp123.iterlim = 0;
solve mcp123 using mcp;
abort$(mcp123.objval > 1e-4) "Benchmark model does not calibrate.";
mcp123.iterlim = 10000;

Parameter report Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mcp123.objval > 1e-4) "Scenario fails to solve.";
$ondotl
report("W","%replacement%","%labormarket%") = 100*(W-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100 * (C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/10-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("FA","%replacement%","%labormarket%") = 100 * (FA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);
report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

* Tariff reform:

** Tariff reform:

tm = 0;

* Consider four alternative closures depending on revenue

* replacement (lumpsum versus wage tax) and labor market

* (flexible versus fixed wages).

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_RS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report
UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;

4.53.1.8.3.4 Model MGENESTED: MPSGE Model with Nesting

$title Static 123 Model Ala Devarjan

$include 123data.gms

parameter
cd0 Final demand for domestic goods
   cm0 Final demand for imports
   sigmac Armington elasticity in final demand /0.5/;

* In this version of the model, we apply the tariff on imports
* in the M block, so we then can measure imports as value gross
* of tariff:
m0 = pm0*m0;

* Store the benchmark tax revenue in the tax rate parameter:
ta = a0 * ta;

* Impute final demand for domestic and imported goods:
cd0 = c0 * d0/(d0+m0);
cm0 = c0 * m0/(d0+m0);

* Armington supply net final demand:
a0 = d0+m0-cd0-cm0+ta;

* Recalibrate taxes on A so that tax revenue remains unchanged:
ta = ta/a0;
$ontext
$model:MGE123

$SECTORS:
Y   ! Production
A   ! Armington composite
M   ! Imports
X   ! Exports

$COMMODITIES:
PD   ! Domestic price index
PX   ! Export price index
PM   ! Import price index
PA   ! Armington price index
PL   ! Wage rate index
RK   ! Rental price index
PFX  ! Foreign exchange

$CONSUMERS:
HH   ! Private households
GOVT ! Government

$AUXILIARY:
TAU_LS   ! Lumpsum Replacement tax
TAU_TL   ! Labor tax replacement
UR       ! Unemployment rate

$PROD:Y  t:etadx s:esubkl
O:PD    Q:d0   P:1   ! YD
O:PX    Q:x0   P:px0  A:GOVT T:tx   ! YX
I:RK    Q:kd0  P:rr0  A:GOVT T:tk   ! KD
I:PL    Q:ly0  P:pl0  A:GOVT T:tl N:TAU_TL   ! LY

$report:
v:YD    o:PD    prod:Y
v:YX    o:PX    prod:Y
v:KD    i:RK    prod:Y
v:LY    i:PL    prod:Y

$PROD:X
O:PFX   Q:(pwx*x0)
I:PX    Q:x0

$PROD:A  s:sigmadm
O:PA    Q:a0   A:GOVT t:ta
I:PD    Q:(d0-cd0)   ! DA
I:PM    Q:(m0-cm0)

$report:
v:DA    i:PD    prod:A
v:MA    i:PM    prod:A

$PROD:M
O:PM    Q:m0
I:PFX   Q:(pwm*m0/pm0) A:GOVT t:tm
$DEMAND: GOVT
E:PFX Q:bopdef
E:PA Q:dtax
E:PA Q:g0 R:TAU_LS
D:PA

$CONSTRAINT: UR
PL = 0 = PA;

$CONSTRAINT: TAU_LS
GOVT = e = PA * g0;

$CONSTRAINT: TAU_TL
GOVT = e = PA * g0;

$DEMAND: HH s: sigma c: sigmac
E:PA Q: (-g0) R: TAU_LS
E:PA Q: (-dtax)
E:RK Q: kd0
E:PA Q: (-i0)
E:PL Q: (ly0+10) ! Labor endowment = ly0+10 - UR * (ly0+10)
E:PL Q: (-ly0+10) R: UR
D:PL Q: i0
D:PD Q: cd0 c:
D:PM Q: cm0 c:

$report:
v: W w: HH
v: CD d: PD demand: HH
v: CM d: PM demand: HH
v: LD d: PL demand: HH

$offtext
$sysinclude mpsgeset mge123
UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP = INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
$include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate."
;
mge123.iterlim = 10000;

parameter report Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve."

report("W", "replacement", "labormarket") = 100*(W.L-1);
report("Y", "replacement", "labormarket") = 100*(Y.L-1);
report("A", "replacement", "labormarket") = 100 * (A.L-1);
report("M", "replacement", "labormarket") = 100 * (M.L-1);
report("X", "replacement", "labormarket") = 100 * (X.L-1);
report("YD", "replacement", "labormarket") = 100 * (YD.L/d0-1);
report("YX", "replacement", "labormarket") = 100 * (YX.L/x0-1);
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1303

report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/(d0-cd0)-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/(m0-cm0)-1);
report("CD","%replacement%","%labormarket%") = 100 * (CD.L/cd0-1);
report("CM","%replacement%","%labormarket%") = 100 * (CM.L/cm0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/10-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);

report(\"TAU_LS\",\"%replacement%\",\"%labormarket%\") = 100*TAU_LS.L;
report(\"TAU_TL\",\"%replacement%\",\"%labormarket%\") = 100*TAU_TL.L;
report(\"UR\",\"%replacement%\",\"%labormarket%\") = 100*UR.L;

* Tariff reform:

tm = 0;
UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

* Lump sum revenue replacement -- downward rigid wage:
UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scdir%report

option report:1:1:2;
display report;

4.53.2 MPSGE Models in GAMS

MPSGE is a mathematical programming system for general equilibrium analysis which operates as a subsystem within GAMS. MPSGE is essentially a library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. MPSGE simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE requires the GAMS/BASE Module including the MILES MCP solver. Optionally it can use the PATH MCP solver.

4.53.2.1 Introduction

This paper introduces a programming language for economic equilibrium modeling. The paper presents the motivation for the system, the programming syntax, and three small scale examples. A library of larger models are provided with the program. The purpose of the paper is to provide a concise introduction to the modeling environment.

MPSGE is a modeling language specially designed for solving Arrow-Debreu economic equilibrium models. (See Rutherford (1987, 1989).) The name stands for "mathematical programming system for general equilibrium". The idea of the MPSGE program is to provide a transparent and relatively painless way to write down and analyze complicated systems of nonlinear inequalities. The language is based on nested constant elasticity of substitution utility functions and production functions. The data requirements for a model include share and elasticity parameters for all the consumers and production sectors included in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

GAMS, the "Generalized Algebraic Modeling System", is a modeling language which was originally developed for linear, nonlinear and integer programming. This language was developed over 15 years ago by Alex Meeraus when he was working at the World Bank. (See Brooke, Kendrick and Meeraus (1988).) Since that time, GAMS has been widely applied for large-scale economic and operations research modeling projects.

MPSGE and GAMS embody different philosophies in their designs. MPSGE is appropriate for a specific class of nonlinear equations, while GAMS is capable of representing any system of algebraic equations. While GAMS is applicable in several disciplines, MPSGE is only applicable in the analysis of economic models within a particular domain. The expert knowledge embodied in MPSGE is of particular use to
economists who are interested in the insights provided by formal models but who are unable to devote many hours to programming. *MPSGE* provides a structured framework for novice modellers. When used by experts, *MPSGE* reduces the setup cost of producing an operational model and the cost of testing alternative specifications.

Prior to the connection with *GAMS*, the "achilles heel" of *MPSGE* had been the process by which input data was translated into the tabular format of the *MPSGE* input file. For small models, this translation was not difficult. Given a calibrated "benchmark equilibrium dataset", a couple of hours with a word processor is usually enough time to generate and investigate a moderately large model. If, however, a model involves several classes of sectors and agents, a wide range of tax instruments and large tables of input data, the word-processor approach is impossible. When there are many numbers, there are many opportunities for oversights and typographical errors.

In contrast, the *GAMS* modeling language is designed for managing large datasets. The use of sets and detached-coefficient matrix notation makes the *GAMS* environment very nice for both developing balanced benchmark datasets and for writing solution reports. For large complicated models, a shortcoming of the *GAMS* modeling environment lies in the specification of the nonlinear equations. Economic equilibrium models, particularly those based on complicated functions such as nested CES, are easier to understand at an abstract level than they are to specify in detail, and the translation of a model from input data into algebraic relations can be a tedious and error-prone undertaking.

The interface between *GAMS* and *MPSGE* combines the strengths of both programs. The system uses *GAMS* as the "front end" and "back end" to *MPSGE* facilitating data handling and report writing. The language employs an extended *MPSGE* syntax based on *GAMS* sets, so that model specification is very concise. In addition, the system includes two large-scale solvers, MILES (Rutherford (1993)) and PATH (Ferris and Dirkse (1993)), which may be used interchangeably. The availability of two algorithms greatly enhances robustness and reliability.

The remainder of this paper is organized as follows. Section 2 introduces *MPSGE* input syntax and the *GAMS* interface using a small two-sector model of taxation. Section 3 extends the 2x2 model to illustrate how the software is used to perform equal-yield (differential) tax policy analysis and to analyze tax reform in a model with endogenous taxation. Section 4 provides a brief summary and conclusion. The paper introduces language features largely through example. Details on language syntax and program structure are provided in two appendices. Appendix A provides a complete statement of *MPSGE* syntax and a summary of differences with the original (1989) language. Appendix B provides an overview of the modeling environment and the structure of *GAMS* input files which employ *MPSGE*.

Before proceeding, both to placate impatient readers and to provide some hands-on experience for novices, I recommend that readers install the *GAMS* system, then retrieve and process the library file THREEMGE which contains three *MPSGE* models (*HARBERGER*, *SHOVEN* and *SAMUELSON*). Two commands to retrieve and run these models:

```
gamslib threemge
```
```
gams threemge
```

After having processed this file, print the listing files (*THREEMGE.LST*) for reference.

### 4.53.2.2 A Mathematical Introduction

Mathiesen (1985) demonstrated that an Arrow-Debreu general economic equilibrium model could be formulated and efficiently solved as a complementarity problem. Mathiesen's formulation may be posed in terms of three sets of "central variables":

---
\( p = \) a non-negative \( n \)-vector of commodity prices including all final goods, intermediate goods and primary factors of production;

\( y = \) a non-negative \( m \)-vector of activity levels for constant returns to scale production sectors in the economy; and

\( M = \) an \( h \)-vector of income levels, one for each "household" in the model, including any government entities.

An equilibrium in these variables satisfies a system of three classes of nonlinear inequalities.

**Zero Profit**

The first class of constraint requires that in equilibrium no producer earns an "excess" profit, i.e. the value of inputs per unit activity must be equal to or greater than the value of outputs. This can be written in compact form as:

\[
- \prod_j(p) = C_j(p) - R_j(p) \geq 0 \quad \forall j
\]

where \( \prod_j(p) \) is the unit profit function, the difference between unit revenue and unit cost, defined as:

\[
C_j(p) = \min \{ \sum_i p_i x_i | f_j(x) = 1 \}
\]

and

\[
R_j(p) = \max \{ \sum_i p_i y_i | g_j(y) = 1 \}
\]

where \( f \) and \( g \) are the associated production functions characterizing feasible input and output. For example, if we have:

\[
f(x) = \phi \prod_i x_i^{\alpha_i} \quad \sum_i \alpha_i = 1, \quad \alpha_i \geq 0
\]

and

\[
g(y) = \psi \max_i \frac{y_i}{\beta_i} \quad \beta_i \geq 0
\]

then the dual functions will be:

\[
C(p) = \frac{1}{\phi} \prod_i \left( \frac{p_i}{\alpha_i} \right)^{\alpha_i}
\]

and

\[
R(p) = \sum_i \beta_i p_i
\]

**Market Clearance**

The second class of equilibrium conditions is that at equilibrium prices and activity levels, the supply of any commodity must balance or exceed excess demand by consumers. We can express these conditions as:

\[
\sum_j y_i \frac{\delta \prod_j(p)}{\delta p_i} + \sum_k \omega_{ik} \geq \sum_k d_{ik}(p, M_k)
\]
in which the first sum, by Shepard’s lemma, represents the net supply of good \( i \) by the constant-returns to scale production sectors, the second sum represents the aggregate initial endowment of good \( i \) by households, and the sum on the right-hand-side represents aggregate final demand for good \( i \) by households, given market prices \( p \) and household income levels \( M \). Final demand are derived from budget-constrained utility maximization:

\[
d_{ih}(p, M_h) = \arg\max \{ U_h(x) \mid \sum_i p_i x_i = M_h \}
\]

in which \( U_h \) is the utility function for household \( h \).

**Income Balance**

The third condition is that at an equilibrium, the value of each agent’s income must equal the value of factor endowments:

\[
M_h = \sum_i p_i \omega_{ih}
\]

We always work with utility functions which exhibit non-satiation, so Walras’ law will always hold:

\[
\sum_i p_i d_{ih} = M_h = \sum_i p_i \omega_{ih}
\]

Aggregating market clearance conditions using equilibrium prices and the zero profit conditions using equilibrium activity levels, it then follows that:

\[
\sum_j y_{ij} \Pi_j(p) = 0
\]

or

\[
y_{ij} \Pi_j(p) = 0 \quad \forall_j
\]

Furthermore, it follows that:

\[
p_i \left( \sum_j y_{ij} \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} - \sum_h d_{ih}(p, M_h) \right) = 0 \quad \forall_i
\]

In other words, complementary slackness is a feature of the equilibrium allocation even though it is not imposed as an equilibrium condition, per-se. This means that in equilibrium, an production activity which is operated makes zero profit and any production activity which earns a negative net return is idle. Likewise, any commodity which commands a positive price has a balance between aggregate supply and demand, and any commodity in excess supply has an equilibrium price of zero.

### 4.53.2.3 A small example: Harberger

This section of the paper introduces MPSGE model building using a two-good, two-factor (2x2) example. This is addressed to readers who may be unfamiliar with GAMS and/or the original (scalar) MPSGE syntax. The discussion provides some details on the formulation and specification of one small model from the MPSGE library. Subsequently, two extensions are presented, one which illustrates equal yield constraints and another which introduces a pure public good. These examples are by no means exhaustive of the classes of equilibrium structures which can be investigated using the software, but they do provide a starting point for new users.

The structure of MPSGE model HARBERGER is “generic” Arrow-Debreu with taxes. Households obtain income by supplying factors of production to industry or collecting tax revenue. This income is then allocated between alternative goods in order to maximize welfare subject to the budget constraint.
Firms operate subject to constant returns to scale, selecting factor inputs in order to minimize cost subject to technological constraints. For an algebraic description of a model closely related to this one, see Shoven and Whalley (1984). The present model differs in two respects from the Shoven-Whalley example. First, in this model there are intermediate inputs to production while in the Shoven-Whalley model goods are produced using only value-added. Second, this model incorporates a labor-leisure choice so that the excess burden of factor taxes here incorporates the disincentive to work associated with a lower net wage.

**Benchmark Data**

Table 1 presents most of the input data for a two good, two factor, closed economy model. This is an economy in which, initially, taxes are levied only on capital inputs to production. We treat tax revenue as though it were returned lump-sum to the households.

<table>
<thead>
<tr>
<th>Sectors</th>
<th>Consumers</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Y OWNERS WORKERS GOVT</td>
<td></td>
</tr>
<tr>
<td>PX 100 -20 -30 -50</td>
<td></td>
</tr>
<tr>
<td>PY -10 80 -40 -30</td>
<td></td>
</tr>
<tr>
<td>PK -20 -40 60</td>
<td></td>
</tr>
<tr>
<td>PL -50 -10 100 -40</td>
<td></td>
</tr>
<tr>
<td>TRNS 10 20 -30</td>
<td></td>
</tr>
<tr>
<td>TK -20 -10 30</td>
<td></td>
</tr>
</tbody>
</table>

The input data is presented in the form of a balanced matrix, the entries in which represent the value of economic transactions in a given period (typically one year). Social accounting matrices (SAMs) can be quite detailed in their representation of an economy, and they are also quite flexible. All sorts of inter-account taxes, subsidies and transfers can be represented through an appropriate definition of the accounts.

Traditionally, a SAM is square with an exact correspondence between rows and columns. (For an introduction, see Pyatt and Round, "Social Accounting Matrices: A Basis for Planning", The World Bank, 1985.) The numbers which appear in a conventional SAM are typically positive, apart from very special circumstances, whereas the rectangular SAM displayed in Table 1 follows a sign convention wherein supplies or receipts are represented by positive numbers and demands or payments are represented by negative numbers. Internal consistency of a rectangular SAM implies that row sums and column sums are zero. This means that supply equals demand for all goods and factors, tax payments equal tax receipts, there are no excess profits in production, the value of each household expenditure equals the value of factor income plus transfers, and the value of government tax revenue equals the value of transfers to households.

With simple MPSGE models, it is convenient to use a rectangular SAM format. This format emphasizes how the MPSGE program structure is connected to the benchmark data. In the rectangular SAM, we have one row for every market (traded commodity). In the present model, there are four markets, for goods X and Y and factors L and K.

There are two types of columns in the rectangular SAM, corresponding to production sectors and consumers. In the present model, there are two production sectors (X and Y) and three consumers (OWNERS, WORKERS and GOVT).

**Data Entry in GAMS**

Consider a generalized version of the model in which the set of production sectors be denoted \( S \) (here, \( S = \{X, Y\} \)). Let the set of goods be \( G \). Production sectors are mapped one-to-one with the goods, so we see that sets \( S \) and \( G \) are in fact the same set. Let \( F \) denote the set of primary factors (here, \( F = L, K \)), and let \( H \) denote the set of households (here \( H = \{\text{OWNER, WORKER}\} \)). Now that we have identified the underlying sets, we may interpret the input matrix as a set of parameters with which we can easily specify the benchmark equilibrium. (See Table 2.) It is quite common to begin a general equilibrium modeling project with a large input-output table or social accounting matrix which may then be mapped onto a number of submatrices, each of which is dimensioned according to the underlying sets used in the model.
The GAMS specification of benchmark data is presented in Table 3 which begins with a statement of the underlying sets \((G,F,H)\). The statement "ALIAS \((S,G)\)"; simply says that \(S\) and \(G\) both reference \(X,Y\). Thereafter follows the social accounting data table and declarations for the various submatrices. The parameters \(ELAS()\) and \(ESUB()\) are elasticities ("free parameters") which can be chosen independently from the benchmark accounts. The parameters \(TF\) and \(PF\) are calibrated tax and reference price arrays which are computed given benchmark factor and tax payments. (In this model, average and marginal tax rates are not distinguished, so the benchmark marginal tax rate is simply the tax payment divided by the net factor income.)

A general equilibrium model determines only relative prices. For purposes of reporting or constructing value-indices, we use Laspeyres quantity index, \(\theta(G)\), the elements of which correspond to shares of aggregate consumer expenditure in the benchmark period.

Table 3: Data Specification in GAMS for the 2x2 Model Harberger

* SECTION (i) DATA SPECIFICATION AND BENCHMARKING

SETS
\(G\) GOODS AND SECTORS /X, Y/,
\(F\) PRIMARY FACTORS /K, L/,
\(H\) HOUSEHOLDS /OWNER, WORKER/;

ALIAS \((S,G)\);

TABLE SAM(*,*) SOCIAL ACCOUNTING MATRIX

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>OWNER</th>
<th>WORKER</th>
<th>GOVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>100</td>
<td>-20</td>
<td>-30</td>
<td>-50</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>-10</td>
<td>80</td>
<td>-40</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>-20</td>
<td>-40</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>-50</td>
<td>-10</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TK</td>
<td>-20</td>
<td>-10</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>TRN</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PARAMETER

\(A(S)\) BENCHMARK OUTPUT
\(B(G,S)\) USE MATRIX (GOODS INPUTS BY SECTOR)
\(C(G,H)\) HOUSEHOLD DEMAND
\(FD(F,S)\) FACTOR DEMAND BY SECTOR
\(E(F,H)\) FACTOR ENDOWMENTS
\(D(F,H)\) FACTOR DEMAND BY HOUSEHOLDS
\(T(F,S)\) TAX PAYMENT BY FACTOR BY SECTOR
\(TRN(H)\) TRANSFER REVENUE
\(ELAS(S)\) ELASTICITY OF SUBSTITUTION IN PRODUCTION
\(ESUB(H)\) ELASTICITY OF SUBSTITUTION IN DEMAND
\(GREV\) BENCHMARK GOVERNMENT REVENUE
\(TF(F,S)\) FACTOR TAX RATE
\(PF(F,S)\) BENCHMARK FACTOR PRICES GROSS OF TAX
\(\theta(G)\) WEIGHTS IN NUMERAIRE PRICE INDEX
\(WBAR(H)\) BENCHMARK WELFARE INDEX;
* EXTRACT DATA FROM THE SOCIAL ACCOUNTING MATRIX:

\[
\begin{align*}
A(S) &= \text{SAM}(S,S); \\
B(G,S) &= \text{MAX}(0, -\text{SAM}(G,S)); \\
C(G,H) &= -\text{SAM}(G,H); \\
FD(F,S) &= -\text{SAM}(F,S); \\
E(F,H) &= \text{SAM}(F,H); \\
D(F,H) &= 0; \\
TRN(H) &= \text{SAM}("TRN",H); \\
T("K",S) &= -\text{SAM}("TK",S);
\end{align*}
\]

* INSTALL "FREE" ELASTICITY PARAMETERS:

\[
\begin{align*}
E("L","WORKER") &= 100; \\
D("L","WORKER") &= 40; \\
\text{ELAS}(S) &= 1; \\
\text{ESUB}(H) &= 0.5;
\end{align*}
\]

* INSTALL FUNCTIONS OF BENCHMARK VALUES:

\[
\begin{align*}
\text{GREV} &= \text{SUM}(H, \text{TRN}(H)); \\
\text{TF}(F,S) &= \text{T}(F,S) / \text{FD}(F,S); \\
\text{PF}(F,S) &= 1 + \text{TF}(F,S); \\
\text{THETA}(G) &= \text{SUM}(H, C(G,H)); \\
\text{THETA}(G) &= \text{THETA}(G) / \text{SUM}(S, \text{THETA}(S)); \\
\text{WBAR}(H) &= \text{SUM}(G, C(G,H)) + \text{SUM}(F, D(F,H));
\end{align*}
\]

Model Specification

The MPSGE description of this model is shown in Table 4. Declarations following the $MODEL statement indicate that the model involves one class of production activities (AL(S)), three classes of commodities (P(G), W(F) and PT), and two types of consumers, private consumers (RA(H)), and a government "consumer" (GOVT).

One $PROD: block describes the single class of production activities, and two $DEMAND: blocks characterize endowments and preferences for the two classes of consumers.

Consider the records associated with production sector AL(S). The entries on the first line of a $PROD: block are elasticity values. The "s:0" field indicates that the top-level elasticity of substitution between inputs is zero (Leontief). The entry "a:ELAS(S)" indicates that inputs identified as belonging to the "a:" aggregate trade off with an elasticity of substitution ELAS(S) (at the second level of the production function). In these production functions, the primary factors (W(F)) are identified as entering in the a: aggregate.

TABLE 4: MPSGE Model Specification and Benchmark Replication

* SECTION (ii) MPSGE MODEL DECLARATION

$ONTXT

$MODEL:HARBERGER

$SECTORS:
   AL(S)

$COMMODITIES:
$CONSUMERS:
   RA(H) GOVT

$PROD: AL(S) s:0 a:ELAS(S)
   D:P(S) Q:A(S)
   I:P(G) Q:B(G,S)
   I:W(F) Q:FD(F,S) P:PF(F,S) A:GOVT T:TF(F,S) a:

$DEMAND: RA(H) s:1 a:ESUB(H)
   D:P(G) Q:C(G,H) a:
   D:W(F) Q:D(F,H)
   E:W(F) Q:E(F,H)
   E:PT Q:TRN(H)

$DEMAND: GOVT
   D:PT Q:GREV

$REPORT:
   V:CD(G,H) D:P(G) DEMAND:RA(H)
   V:DF(F,H) D:W(F) DEMAND:RA(H)
   V:EMPLOY(S) I:W("L") PROD:AL(S)
   V:WLF(H) W:RA(H)

$OFFTEXT

* Invoke the preprocessor to declare the model for GAMS:

$SYSINCLUDE mpsgeset HARBERGER

* ------------------------------------------------------------------

* SECTION (iii) BENCHMARK REPLICATION

HARBERGER.ITERLIM = 0;
$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;
ABORT$(ABS(HARBERGER.OBJVAL) GT 1.E-4)
   "*** HARBERGER benchmark does not calibrate."
HARBERGER.ITERLIM = 1000;

The records within a $PROD: block begin with "O:" or "I:". An "O:" indicates an output and an "I:" represents an input. In both types of records, "Q:" is a "quantity field" indicating a reference input or output level of the named commodity. A "P:" signifies a reference price field. This price is measured as a user cost, gross of applicable taxes. The default values for reference price and reference quantity are both unity (i.e., a value of 1 is installed if a P: or Q: field is missing).

The A: and T: fields in a $PROD: block indicate tax agent and ad-valorem tax rate, respectively. The tax agent is specified before the tax rate. A single input or output coefficient may have two or more taxes applied. Consumers are treated symmetrically, and there is thus no restriction on the consumer to whom the tax is paid. Typically, however, one consumer is associated with the government.

To better understand the relationship between reference prices and tax rate specification, consider inputs of W.K to sector AL.X in this model. The benchmark payment to capital in the X sector is 20 and the tax payment is 20. Hence the ad-valorem tax rate in the benchmark equilibrium is 100% (T: 1), and the reference price of capital, market price of unity times (1 + 100%), is 2 (P: 2). If in a counterfactual experiment the tax rate on capital inputs to sector X is altered, this will change the T: field but it will
not change the $P: field. $Q: and $P: characterize a reference equilibrium point, and these are therefore unaffected by subsequent changes in the exogenous parameters.

It is important to remember that the $PROD:AL(S)$ block represents as many individual production functions as there are elements in set $S$ (two in this case). Within the $PROD:AL(S)$ block, inputs refer to sets $G$ and $F$, while the output coefficient, $O : P(S)$, refers only to set $S$. Sets referenced within a commodity name in an $I :$ or $O :$ field may be sets which are ”controlled” by the sets referenced in the function itself, in which case only a scalar entry is produced. In $PROD:AL(S)$ there are only outputs of commodity $S$ in sector $S$.

The first line of a $DEMAND$ block also contains fields (e.g., $a:,$ $b:,$ etc.) which represent elasticities of substitution. The subsequent records may begin with either an $E :$ field or a $D :$ field. These, respectively, represent commodity endowments and demands. In the demand fields, the $P :$ and $Q :$ entries are interpreted as reference price and reference quantity, analogous to the input fields in a $PROD$ block. Ad-valorem taxes may not be applied on final demands, so that if consumption taxes are to be applied in a model they must be levied on production activities upstream of the final demand.

The benchmark values for all activity levels and prices are equal to the default value of unity, and therefore we are able to specify values in the $Q :$ fields directly from the benchmark data. An equivalent model could be specified in which the benchmark activity levels for $AL(S)$ equal, for example, $A(S,S)$. This would then require rescaling the input and output coefficients for those sectors, and it would not necessarily be helpful, because in a scaled model it is more difficult to verify consistency of the benchmark accounts and MPSGE input file. Furthermore, for numerical reasons it is advisable to scale equilibrium values for the central variables to be close to unity.

Government transfers to households are accomplished through the use of an ”artificial commodity” (PT). The government is identified as the agent who receives all tax revenue (see the $A:GOVT$ entry in both of the $PROD :$ blocks). Commodity PT is the only commodity on which GOVT spends this income, hence government tax revenue is divided between the two households in proportion to their endowments of the artificial good. In order to scale units so that the benchmark price of PT is unity, the $30 of government tax revenue chases 10 units of PT assigned to OWNER and 20 units assigned to WORKER. (See values for $TRN(H)$ in Table 3.)

The $REPORT$ section of the input file requests the solution system to return values for inputs, outputs, final demands or welfare indices at the equilibrium. Only those items which are requested will be written to the solution file. Each record in the report block begins with a $V :$ (variable name) field. These names must be distinct from all other names in the model. The second field of the report record must have one of the labels $I :,$ $O :$ or $D :$ followed by a commodity name, or the label $W :$ followed by a consumer name. The third field’s label must be $"PROD:"$ in an $I :$ or $O :$ record, and it must be $"DEMAND:"$ if it is a $D :$ record.

An Algebraic Formulation of Harberger Model

The unit cost of production in sector $A_h$ is given by a nested Leontief-CES function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. In equilibrium, the unit cost must be no less than the market price of output:

**Zero Net Profit**

\[- \Pi_s = \sum_g p_g B_{bs} + \phi_s \left( \sum_f \alpha \omega_f (1 + t_f) \right)^{1 - \sigma_s} - p_s \geq 0 \quad \forall s | \sigma_s \neq 1\]

and

\[- \Pi_s = \sum_g p_g B_{bs} + \phi_s \left( \Pi_f (1 + t_f) \right)^{\alpha_f} - p_s \geq 0 \quad \forall s | \sigma_s = 1\]
Income Balance for Government

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates:

\[ PT = \sum_s \frac{\partial \Pi_s}{\partial (\omega_f (1+ t_f s))} AL_s \omega_f t_f s \]

Income Balance for Households

Household income is determined by the net of tax return to primary factors plus the household share of government revenue:

\[ M_h = \sum_f \omega_f E_{fh} + s_h PT \]

Market Clearance for Goods

Producer output is equal to the sum of intermediate plus final demand:

\[ AL_g \geq \sum_s AL_s B_{gs} + \sum_h \frac{\gamma_h M_h}{e_h(p)} \left( \frac{\epsilon_h(p)}{p_g} \right)^{\frac{1}{1-\sigma_h}} \]

where \( \gamma_h \) is the household budget share devoted to the consumption of goods, and \( e_h \) is the “unit expenditure function” which may be written:

\[ e_h(p) \equiv \left( \sum_i \beta_{ih} p_i^{1-\sigma_h} \right)^{\frac{1}{1-\sigma_h}} \]

Market Clearance for Factors

The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be “leisure” or “household production”. Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

\[ \sum_h E_{fh} = \sum_s AL_s \frac{\partial \Pi_s}{\partial (\omega_f (1+ t_f s))} + \sum_h \frac{\mu_h M_h}{\omega_f} \]

MPSGE Formulation: Key Ideas

There are two points regarding the MPSGE function format which are important yet easily misunderstood by new users:

1. The elasticities together with the reference quantities and reference prices of inputs and outputs completely characterize the underlying nested CES functions. No other data fields in the $PROD$ block alters the technology. If, for example, a tax rate changes as part of a counter-factual experiment, this has no effect on the reference price. The value in the $P$ : field depends on the benchmark value of the $T$ : field if the model has been calibrated, but subsequent changes in $T$ : do not change the underlying technology.

2. Tax rates are interpreted differently for inputs and outputs. The tax rate on inputs is specified on a net basis, while the tax rate on outputs is specified on a gross basis. That is, the user cost of an input with market price \( p \) subject to an ad-valorem tax at rate \( t \) is \( p(1 + t) \), while the user cost of an output subject to an ad-valorem tax at rate \( t \) is \( p(1 - t) \). (A tax increases the producer cost of inputs and decreases the producer value of outputs.)
MPSGE provides a limited number of economic components with which complex models can be constructed. There are some models which lie outside the MPSGE domain, but in many cases it is possible to recast the equilibrium structure in order to produce an MPSGE model which is logically equivalent to the original model - usually after having introduced some sort of artificial commodity or consumer. In the present model, the use of commodity PT to allocate government revenue between households provides a fairly typical example of how this can be done. In the process of making such a transformation, one often gains a meaningful economic insight.

The Solution Listing

The detailed solution listing for model HARBERGER is shown in Table 5. The standard GAMS report facilities display solution values. Central variables are always either fixed (upper = lower), or they are non-negative (lower bound = 0, upper bound = +INF). The MARGINAL column in the solution report returns the value of the associated slack variable. Complementarity implies that in equilibrium, either the level value of a variable will be positive or the marginal value will be positive, but not both.

The output file (not shown) also provides details on the computational process. For an explanation of these statistics, see Rutherford (1993).

**TABLE 5: GAMS Solution Listing for Model HARBERGER**

<table>
<thead>
<tr>
<th>---</th>
<th>VAR</th>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>1.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>1.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>---</th>
<th>VAR</th>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>1.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>1.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>---</th>
<th>VAR</th>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>X.OWNER</td>
<td></td>
<td>30.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X.WORKER</td>
<td></td>
<td>50.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y.OWNER</td>
<td></td>
<td>40.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y.WORKER</td>
<td></td>
<td>30.000</td>
<td>+INF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Computing Counter-factual Scenarios

Table 6 presents the GAMS code for specification and solution of three counterfactual equilibria. In these experiments, the nonuniform system of capital taxes from the benchmark is replaced by three alternative uniform factor tax structures: a tax on labor, a tax on capital, and a tax on both labor and capital. In each case, the tax rate is chosen to replace the benchmark tax revenue at benchmark prices and demand (ignoring induced substitution effects). Following each solution, the equilibrium values for tax revenue, welfare (Hicksian equivalent variation), employment, prices and output are stored in parameter REPORT.

**TABLE 6: Specification and Processing of Counter-Factual Scenarios**

* * * *

--- VAR DF

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>K.OWNER</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
</tr>
<tr>
<td>K.WORKER</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
</tr>
<tr>
<td>L.OWNER</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
</tr>
<tr>
<td>L.WORKER</td>
<td>.</td>
<td>40.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

--- VAR EMPLOY

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>.</td>
<td>50.000</td>
<td>+INF</td>
</tr>
<tr>
<td>Y</td>
<td>.</td>
<td>10.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

--- VAR WLF

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER</td>
<td>.</td>
<td>1.000</td>
<td>+INF</td>
</tr>
<tr>
<td>WORKER</td>
<td>.</td>
<td>1.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

**** REPORT SUMMARY : 0 NONOPT
0 INFEASIBLE
0 UNBOUNDED
0 ERRORS

* SPECIFY COUNTER-FACTUAL TAX RATES TO ACHIEVE CETERIS
* PARIBUS BALANCED BUDGET:

TAXRATE("L",S,"L") = GREV / SUM(G, FD("L",G));
TAXRATE("K",S,"K") = GREV / SUM(G, FD("K",G));
TAXRATE("L",S,"VA") = GREV / SUM((F,G), FD(F,G));
TAXRATE("K","S","VA") = GREV / SUM((F,G), FD(F,G));

LOOP(SC,
 * INSTALL TAX RATES FOR THIS COUNTERFACTUAL:
   TF(F,S) = TAXRATE(F,S,SC);

$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;

* SECTION (v) REPORT WRITING:

* REPORT SOME RESULTS:

   PINDEX = SUM(G, P.L(G) * THETA(G));

   REPORT("REVENUE","_",SC) = 100 * (PT.L/PINDEX - 1);
   REPORT("TAXRATE","_",SC) =
      100 * SMAX((F,S), TAXRATE(F,S,SC));
   REPORT("WELFARE",H,SC) = 100 * (WLF.L(H) - 1);
   REPORT("EMPLOY",S,SC) = 100 * (EMPLOY.L(S)/FD("L",S) - 1);
   REPORT("PRICE",G,SC) = 100 * (P.L(G)/PINDEX - 1);
   REPORT("PRICE",F,SC) = 100 * (W.L(F)/PINDEX - 1);
   REPORT("OUTPUT",S,SC) = 100 * (AL.L(S) - 1);
);
DISPLAY REPORT;

4.53.2.4 Alternative models: Shoven and Samuelson

The "standard" MPSGE model is based on fixed endowments and tax rates, but many empirical models
do not fit into this structure. For example, in the model HARBERGER, the level of each replacement tax
was specified to be consistent with "equal yield", but as a result of the endogenous response of prices
and quantities, the resulting tax revenues differed significantly from the benchmark levels. For example,
when the capital tax is replaced by a uniform labor tax at a rate which, in the absence of labor supply
response, produces "equal yield", we find that tax revenue in fact declines by 39%. In order to perform
differential (equal yield) tax policy analysis, it is therefore necessary to accommodate the endogenous
determination of tax rates as part of the equilibrium computation. This is one of many possible uses of
"auxiliary variables" in MPSGE.

Tax Analysis with Equal Yield

Table 7 presents the MPSGE model definition for test problem SHOVEN. This model is equivalent to
the HARBERGER, apart from the addition of an auxiliary variable TAU. Within MPSGE, auxiliary variables
can either represent price-adjustment instruments (endogenous taxes) or they can represent a quantity-
adjustment instruments (endowment rations). In model SHOVEN, TAU is used to proportionally scale factor
taxes in order to achieve a target level of government revenue. The auxiliary variable first appears in
the $PROD:AL(S) block, following the declaration of a tax agent. There are two fields associated with an
endogenous tax. The first field (N:) gives the name of the auxiliary variable which will scale the tax rate.
The second field (M:) specifies the multiplier. If the $M: field is omitted, the multiplier assumes a default
value of unity. If the value in the $M: field is zero, the tax does not apply.

The auxiliary variable TAU also appears at the bottom of the file where it labels an associated inequality
constraint. This constraint exhibits complementary slackness with the associated non-negative auxiliary
variable (i.e., if TAU is positive, the constraint must hold with an equality, whereas if the constraint is
non-binding TAU must be zero). An auxiliary variable may or may not appear in its associated constraint.
The constraint associated with $\text{TAU}$ is based on a price index defined by $\text{THETA(G)}$. The constraint assures a level of tax revenue such that the value of transfers to households is held constant. (Endowments of the commodity $\text{PT}$ are fixed, so when the value of $\text{PT}$ is fixed, then so too are the value of transfers from $\text{GOVT}$ to each of the households.)

\textit{SHOVEN} illustrates how an auxiliary variable can be interpreted as a tax instrument. In the \textit{MPSGE} syntax, auxiliary variables may also be employed to endogenously determine commodity endowments. There is no restrictions on how a particular auxiliary variable is to be interpreted. A single variable could conceivably serve simultaneously as an endogenous tax as well as a endowment ratio, although this would be rather unusual.

\textbf{TABLE 7: Differential Tax Policy Analysis}

An Algebraic Formulation of Shoven Model

Zero Net Profit
The unit cost of production in sector $A_s$ is given by a nested Leontief-CES function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. Unlike the Harberger models, tax rates in this model are determined endogenously. In equilibrium, the unit cost must be no less than the market price of output:

$$-\Pi_s = \sum_g p_g B_{gs} + \phi_s \left( \sum_f \alpha_{fs} (\omega_f (1 + \tau t_{fs}))^{1 - \sigma_s} \right)^{1/1 - \sigma_s} - p_s \geq 0 \quad \forall s | \sigma_s \neq 1$$

and

$$-\Pi_s = \sum_g p_g B_{gs} + \phi_s \left( \prod_f (\omega_f (1 + \tau t_{fs}))^{\alpha_{fs}} \right) - p_s \geq 0 \quad \forall s | \sigma_s = 1$$

**Income Balance for Government**

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates:

$$PT = \sum_s \frac{\partial \Pi_s}{\partial (\omega_f (1 + \tau t_{fs}))} AL_s \omega_f \tau t_{fs}$$

**Equal Yield**

In equilibrium, tax rates are multiplicatively adjusted to achieve a target level of government revenue:

$$PT = \sum_g \theta_g p_g$$

**Income Balance for Households**

Household income is determined by the net of tax return to primary factors plus the household share of government revenue:

$$M_h = \sum_f \omega_f E_{fh} + \delta_h PT$$

**Market Clearance for Goods**

Producer output is equal to the sum of intermediate plus final demand:

$$AL_g \geq \sum_s AL_s B_{gs} + \sum_h \gamma_h M_h \left( \frac{\epsilon_h(p)}{p_h} \right)^{\sigma_h}$$

where $\gamma_h$ is the household budget share devoted to the consumption of goods, and $\epsilon_h$ is the "unit expenditure function" which may be written:

$$\epsilon_h(p) \equiv \left( \sum_i \beta_{ih} p_i^{1 - \sigma_h} \right)^{1/1 - \sigma_h}$$

**Market Clearance for Factors**
The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be “leisure” or “household production”. Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

\[
\sum_h E_{fh} = \sum_s AL_s \frac{\partial \Pi_s}{\partial (\omega_f (1 + \tau_t s))} + \sum_h \frac{\mu_h M_h}{\omega_f}
\]

Public Goods and Endogenous Taxation

Consider a final extension of the 2x2 model in which tax revenue funds a pure public good. Model SAMUELSON presented in Table 8. This model illustrates one of several ways that public goods can be modelled in MPSGE. Here the level of public provision is determined by a Samuelson-condition equaling the sum of individual marginal rates of substitution (marginal benefit) with the marginal rate of transformation (marginal cost). Unlike the equal yield formulation, the tax revenues collected by GOVTare not returned lump-sum but are instead used to finance provision of a pure public good. This representation of government has not been widely adopted in the CGE literature, perhaps because of the difficulties involved in specifying preferences for public goods.

The relevant characteristic of a pure public good entering final demand is that each consumer “owns” the same quantity. Agents’ attitudes toward public goods differ, and because there is no market, agents’ valuations of the public good will also differ. In an MPSGE model, the separate valuations are accommodated through the introduction of “personalized” markets for public good - one market for each consumer. In the model, consumer expenditure encompasses both private and public “purchases”, and consumer income encompasses both private and public ”endowments”. An individual is endowed with a quantity of her own version of the public good determined by the level of public expenditures. An increase in taxes, to the extent that it increases tax revenue, will increase the level of public provision.

In this model, the structure of relative factor taxes is exogenous but the aggregate level of taxes is not. Tax rates are scaled up or down so that the sum of individual valuations of the public good (the marginal benefit) equals the cost of supply of the public good (the direct marginal cost).

Consider features of model SAMUELSON which do not appear in SHOVEN:

1. There are new commodities PG and VG(H). The first of these represents the direct marginal cost of public output from sector GP, a Leontief technology which converts private goods inputs into the public good. For the SAMUELSON structure, all government revenues apply to purchases of the public good (observe that the only good demanded by consumer GOVTis PG ). The prices VG(H) represent the individual consumer valuations of the public good. Commodity VG(H) appears only in the endowments and demands of consumer RA(H). The endowment record for VG(H) includes a quantity V(H) which is the benchmark valuation of the public good by agent H.

2. There are two auxiliary variables. TAU has the same interpretation as in the SHOVEN, determining the aggregate tax level. Auxiliary variable LGP is a rationing instrument representing an index of the “level of public goods provision”, scale to equal 1 in the benchmark. Consumer RA(H) thus is endowed with a quantity of VG(H) given by V(H) * LGP.

3. The constraint for TAU in SAMUELSON differs from the TAU constraint in SHOVEN. Here the constraint represents the Samuelson condition, equating the marginal cost (PG*GREV) and the sum of individuals' marginal benefit (SUM(H, VG(H) + V(H))). The constraint for LGP simply assigns LGP equal to the sector GP activity level. (The LGP variable and constraint are only needed because the R: field only accepts auxiliary variables.)

| TABLE 8: Endogenous Determination of Tax Revenue |
An Algebraic Formulation of Samuelson Model

Zero Net Profit for Private Production

The unit cost of production in sector $A_h$ is given by a nested Leontief - Cobb-Douglas function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. Unlike the Harberger models, tax rates in this model are determined endogenously. In equilibrium, the unit cost must be no less than the market price of output:
\[-\Pi_s = \sum_g p_g B_{gs} + \phi_s \left( \prod_f (\omega_f(1 + \tau f_s))^\alpha f_s \right) - p_s \geq 0 \quad \forall s\]

**Zero Net Profit for Public Sector Contractors**

The unit cost of public provision is determined by the market price of commodity inputs to the Leontief activity. Input requirements are defined by a vector of public sector input coefficients, \(A_h g\). In equilibrium, the price paid by the government equals the cost of market inputs:

\[\sum_g p_g a_g = p_G \geq 0\]

**Income Balance for Government**

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates. In equilibrium, the value of tax revenue equals the market cost of public sector output:

\[PT = \sum_s \frac{\partial \Pi_s}{\partial (\omega_f(1 + \tau f_s))} AL_s \omega_f \tau f_s = p_G G\]

**Income Balance for Households**

Household income is determined by the net of tax return to primary factors plus the imputed value of public provision:

\[M_h = \sum_f \omega_f E_{fh} + v_h G\]

**Market Clearance for Private Goods**

Producer output is equal to the sum of intermediate plus final demand:

\[AL_g \geq \sum_s AL_s B_{gs} + \sum_h \gamma_h M_h \left( \frac{e_h(p)}{p_g} \right)^{\sigma_h}\]

where \(\gamma_h\) is the household budget share devoted to the consumption of goods, and \(e_h(p)\) is the “unit expenditure function” which may be written:

\[e_h(p) \equiv \left( \sum_i \beta_i h P_i^{1-\sigma_h} \right)^{-\frac{1}{\sigma_h}}\]

**Personalized Markets for Public Goods**

We assume a “pure” public good in this model, hence each household may attach a different marginal valuation to public provision in an equilibrium. In order to compute these marginal values, we include a separate public good “market” for each household which balances the level of provision with the household “demand”:

\[G = \frac{\mu_G M_h}{v_h}\]
in which \( \mu_{G,h} \) is the budget share of public goods in the top-level Cobb-Douglas preferences of household \( h \).

**Market Clearance for Factors**

The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be “leisure” or “household production”. Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

\[
\sum_h E_{fh} = \sum_s AL_s \frac{\prod_{t} \omega_{f(t+1)}}{\omega_f} + \sum_h \mu_{f,h} M_h \omega_f
\]

**Samuelson Rule for “Optimal” Provision of Public Goods**

The tax rate multiplier is adjusted to balance the marginal cost of public provision with the summation across households of marginal willingness to pay. Due to the existence of household factor demand, factor taxes are necessarily distortionary and there will be an excess social cost of public funds. For this reason, the Samuelson rule is neither necessary nor sufficient for optimal provision. We apply the rule here merely to illustrate the programming methodology, even though the resulting equilibrium may be “suboptimal”:

\[
PG = \sum_h v_h
\]

**Comparing Model Results**

Although the foregoing discussion has focused on the nuances of MPSGE model syntax, but there are many interesting economic questions which can be addressed using even small-scale models such as those described here. Consider the output listing from parameter REPORT is displayed in Table 9. It is perhaps surprising to note that none of the uniform tax structures represents a Pareto-superior choice compared to the benchmark tax structure. Furthermore, from the standpoint of aggregate welfare ("WELFARE.TOTAL" = income-weighted sum of individual EV's), only the uniform capital tax represents an improvement.

**Table 9: Numerical Results from Alternative Models**

<table>
<thead>
<tr>
<th>INDEX 1 = HARBERGER</th>
<th>K</th>
<th>L</th>
<th>VA</th>
</tr>
</thead>
<tbody>
<tr>
<td>REVENUE._</td>
<td>3.9</td>
<td>-38.9</td>
<td>-0.8</td>
</tr>
<tr>
<td>TAXRATE._</td>
<td>50.0</td>
<td>50.0</td>
<td>25.0</td>
</tr>
<tr>
<td>WELFARE.OWNER</td>
<td>1.9</td>
<td>42.4</td>
<td>18.5</td>
</tr>
<tr>
<td>WELFARE.WORKER</td>
<td>-0.1</td>
<td>-26.8</td>
<td>-10.9</td>
</tr>
<tr>
<td>WELFARE.TOTAL</td>
<td>0.6</td>
<td>-1.3</td>
<td>-3.48143E-2</td>
</tr>
<tr>
<td>EMPLOY .X</td>
<td>-5.3</td>
<td>-6.9</td>
<td>-8.4</td>
</tr>
<tr>
<td>EMPLOY .Y</td>
<td>20.5</td>
<td>34.4</td>
<td>22.1</td>
</tr>
<tr>
<td>PRICE .X</td>
<td>-10.4</td>
<td>-11.2</td>
<td>-10.3</td>
</tr>
<tr>
<td>PRICE .Y</td>
<td>11.8</td>
<td>12.8</td>
<td>11.8</td>
</tr>
<tr>
<td>PRICE .K</td>
<td>3.9</td>
<td>59.5</td>
<td>24.5</td>
</tr>
<tr>
<td>PRICE .L</td>
<td>-4.7</td>
<td>-38.9</td>
<td>-23.5</td>
</tr>
<tr>
<td>OUTPUT .X</td>
<td>3.6</td>
<td>-1.0</td>
<td>0.4</td>
</tr>
<tr>
<td>OUTPUT .Y</td>
<td>-3.7</td>
<td>2.0</td>
<td>-2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INDEX 1 = SHOVEN</th>
<th>K</th>
<th>L</th>
<th>VA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAXRATE._</td>
<td>47.1</td>
<td>134.2</td>
<td>25.3</td>
</tr>
<tr>
<td>WELFARE.OWNER</td>
<td>3.3</td>
<td>40.2</td>
<td>18.3</td>
</tr>
<tr>
<td>WELFARE.WORKER</td>
<td>-1.0</td>
<td>-29.2</td>
<td>-10.8</td>
</tr>
<tr>
<td>WELFARE.TOTAL</td>
<td>0.6</td>
<td>-3.6</td>
<td>-3.51710E-2</td>
</tr>
<tr>
<td>EMPLOY .X</td>
<td>-5.0</td>
<td>-19.7</td>
<td>-8.5</td>
</tr>
</tbody>
</table>
### 4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1323

**Employment**

\[
\begin{array}{ccc}
\text{EMPLOY .Y} & 21.5 & 12.1 & 21.9 \\
\text{PRICE .X} & -10.4 & -9.0 & -10.3 \\
\text{PRICE .Y} & 11.9 & 10.2 & 11.8 \\
\text{PRICE .K} & 6.2 & 49.8 & 24.2 \\
\text{PRICE .L} & -5.0 & -56.5 & -23.6 \\
\text{OUTPUT .X} & 3.6 & -7.9 & 0.3 \\
\text{OUTPUT .Y} & -3.4 & -2.0 & -2.1 \\
\end{array}
\]

**Index 1 = SAMUELSON**

\[
\begin{array}{ccc}
\text{REVENUE .} & -1.4 & -14.5 & -6.7 \\
\text{TAXRATE .} & 45.7 & 88.8 & 22.8 \\
\text{WELFARE .OWNER} & 4.7 & 43.9 & 21.1 \\
\text{WELFARE .WORKER} & -2.0 & -31.4 & -12.9 \\
\text{WELFARE .TOTAL} & 0.5 & -3.7 & -0.4 \\
\text{EMPLOY .X} & -4.9 & -7.5 & -5.9 \\
\text{EMPLOY .Y} & 24.5 & 37.5 & 29.7 \\
\text{PRICE .X} & -10.7 & -11.3 & -10.9 \\
\text{PRICE .Y} & 12.2 & 13.0 & 12.5 \\
\text{PRICE .K} & 7.8 & 60.3 & 29.0 \\
\text{PRICE .L} & -6.0 & -51.8 & -24.5 \\
\text{OUTPUT .X} & 3.0 & -2.2 & 0.9 \\
\text{OUTPUT .Y} & -2.3 & 3.3 & -2.58148E-2 \\
\text{PROVISION .} & -0.8 & -13.9 & -6.1 \\
\end{array}
\]

#### 4.53.2.5 Summary

This paper has provided an introduction to a new GAMS subsystem for applied general modeling. This extension of GAMS accommodates a tabular representation of highly nonlinear cost and expenditure functions through which model specification is concise and transparent. The paper has presented three small examples which illustrate the programming environment and its application to traditional economic issues in public finance for which applied general equilibrium analysis is a standard tool. Further work is underway in the development and evaluation of solution algorithms for applied general equilibrium models implemented within GAMS/MPSGE. In addition to providing a convenient framework for model-builders, the new GAMS/MPSGE system also simplifies the implementation and testing of algorithms for complementarity problems. Information on the relative effectiveness of different solution strategies should prove quite helpful to users who are using the system to solve very large systems of nonlinear equations.

#### 4.53.2.6 References

4.53.2.7 Appendix A: Language Syntax

General syntax rules

- All input is free format (spaces and tabs are ignored) except keywords for which "$" must appear in column 1.
- End-of-line is significant. Continuation lines are indicated by a "+" in column 1.
- In general, input is not case sensitive, except in the specification of sub-nests for production and demand functions.
- Numeric expression involving GAMS parameters or constants must be enclosed in parentheses.

Keywords

Keywords typically appear in the following order:

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ONTENT</td>
<td>Indicate the beginning of a GAMS comment block containing an MPSGE model.</td>
</tr>
<tr>
<td>$MODEL: model_name</td>
<td>model_name must be a legitimate file name. This name is subsequently used</td>
</tr>
<tr>
<td></td>
<td>to form MODEL_NAME.GEN (this file name must be upper case when running</td>
</tr>
<tr>
<td></td>
<td>under UNIX).</td>
</tr>
<tr>
<td>$SECTORS:, $COMMODITIES:, $AUXILIARY:, $CONSUMERS:</td>
<td>Four keywords define variables which are used in the model. Entries in these blocks share the same syntax. The $AUXILIARY block is only used in models with side constraints and endogenous taxes or rationed endowments.</td>
</tr>
<tr>
<td>$PROD: sector</td>
<td>Production functions must be specified for each production sector in the model.</td>
</tr>
<tr>
<td>$DEMAND: consumer</td>
<td>Demand functions must be specified for every consumer in the model. General structure is the same as for production functions above.</td>
</tr>
<tr>
<td>$CONSTRAINT: auxiliary</td>
<td>Specifies a side constraint to be associated with a specified auxiliary variable.</td>
</tr>
<tr>
<td>$REPORT:</td>
<td>Identifies the set of additional variables to be calculated. These include outputs and inputs by sector and demands by individual consumers.</td>
</tr>
<tr>
<td>$OFFTEXT</td>
<td>Indicates the end of model specification.</td>
</tr>
</tbody>
</table>

Variable Declarations

There are four classes of variables within an MPSGE model: activity levels for production sectors, prices for commodities, income levels for consumers and level values for auxiliary variables. These classes of variables are distinguished in order to permit additional semantic checking by the MPSGE preprocessor. For example, if P has been declared as a price (within the $COMMODITIES: block), then the preprocessor would report an error if it encountered "$PROD:P".

The $SECTORS:, $COMMODITIES:, $CONSUMERS: and $AUXILIARY blocks contain implicit GAMS variable declarations in which the index sets must be specified in the GAMS program above and the variable names must be distinct from all other symbols in the GAMS program. One or more variables may be declared per line separated by one or more spaces.
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1325

$SECTORS:

Y(R,T) ! Output in region R in period T
K(T) ! "Aggregate capital stock, period T"

In these declarations, the trailing comments (signified by "!") are interpreted as variable name descriptors which subsequently appear in the solution listing.

The equivalent GAMS declaration for these variables would be:

VARIABLES Y(R,T) Output in region R in period T
K(T) "Aggregate capital stock, period T";

As with the usual GAMS syntax, when a variable descriptor contains a punctuation symbol such as ",", it is required to enclosed in quotes.

$SECTORS:

X(R,T)$X0(R) GT 0)

Here, the GAMS conditional operator "$" is used to restrict the domain of the variable X. The expression following the dollar sign is passed through to the GAMS compiler and must conform to GAMS syntax rules.

$SECTORS:

X Y(R)$Y0(R) Z ! Descriptor for Z
W(G,R,T) ! Descriptor for W

More than one symbol may appear on a single line. The descriptor only applies to the last one.

All MPSGE variables must be declared. When multidimensional variables are specified, they must be declared explicitly - declarations like X(∗) are not permitted. Two further restrictions are that the sets used in the declaration must be static rather than dynamic, and any variable which is declared must be used in the model. There is a simple way to work around these restrictions. Let me illustrate with an example. Suppose that in a model the set of production sectors AL is employed for all elements of a static set S which satisfy a particular condition, for example BMX(S) not equal to 0. This would require that AL be declared as follows:

$SECTORS:

AL(S)$BMX(S)

In this context, the symbol "$" is used as an "exception operator" which should be read as "such that". In this case, we have generated one AL sector for each element of the set S for which BMX(S) is nonzero.

Function Declarations

Functional declarations characterize nested CES functions which characterize both preferences and technology. The former are written within a $DEMAND: block and the latter within a $PROD: block. Tax entries may appear within a $PROD: block but not within a $DEMAND: block, otherwise the syntax is nearly identical. The syntax for these blocks will be described through a sequence of examples:

$PROD:Y(R) s:1
0:P(R) Q:Y0(R)
I:W(F,R) Q:FDO(F,R)
This block characterizes a Cobb-Douglas production function in which the elasticity of substitution between inputs is one - "s:1" in the first line which sets a top level substitution elasticity equal to unity.

Variable Y(R) is an activity level declared within the $SECTORS: block. Variables P(R) and W(F,R) are prices declared within the $COMMODITIES: block. The O: label indicates an output, and the I: prefix indicates an input. The Q: fields in both records represent "reference quantities". Y0(R) and F00(F,R) must be GAMS parameters defined previously in the program.

$$PROD:X(R) \quad s:ESUB(R) \quad a:0 \quad b: (ESUB(R)*0.2)$$

The keyword line specifies three separate elasticities related to this function. ESUB(R) is the top level elasticity of substitution. There are two sub-nests in the function. Nest a: is a Leontief nest (in which the compensated elasticity is zero). The elasticity of substitution in nest b: is one-fifth of the top level elasticity.

In the function specification, commodities PY(G) (one input for each element of set G) enter in fixed proportions. Commodities PL and PK enter in nest b.

If this function has been specified using a balance benchmark dataset with reference prices equal to unity, then the following identity should be satisfied:

$$X0(R) = \text{SUM}(G, YX0(G,R)) + LX0(R) + KX0(R)$$

$$PROD:AL(S) \quad s:0 \quad a:ELAS(S)$$

In this function, we have two new ideas. The first is the use of a reference price denoted by "P:". This entry indicates that the function should be calibrated to a reference point where individual input prices (gross of tax) equal PF(F,S). If P: does not appear, prices of one are assumed.

The second new idea here is that taxes may be levied on production inputs. The A: label identifies the name of the tax agent (a $CONSUMER:). The T: label identifies the ad-valorem tax rate.

$$DEMAND:RA(R)\quad RA0(R) \quad s:1$$

This function specification demonstrates the use of conditionals. This function is only generated when RA0(R) is nonzero. The demands D: for a particular element of set G are suppressed entirely when DG(G) equals 0. The Q: field also has an exception operator, so that the default value for Q: (unity) is applied when DD(G,R) equals zero.

This example is somewhat artificial, but it illustrates the distinction between how exception operators affect lead entries (I:, O:, D: and E:) and subsequent entries. When an exception is encountered on the lead entry, the entire record may be suppressed. Exceptions on subsequent entries only applied to a single field.

The valid labels in a function declaration ($PROD: or $DEMAND: ) line include:
Mathematical Programming System for General Equilibrium analysis (MPSGE)

\( s \): Top level elasticity of substitution between inputs or demands.

\( t \): Elasticity of transformation between outputs in production. (valid only in $PROD$ blocks)

\( a;,b;,… \) Elasticities of substitution in individual input nests.

The recognized labels in an \( I; \) or \( O; \) line include:

- \( Q \): Reference quantity. Default value is 1. When specified, it must be the second entry.
- \( P \): Reference price. Default value is 1.
- \( A \): Tax revenue agent. Must be followed by a consumer name.
- \( T \): Tax rate field identifier. (More than one tax may apply to a single entry.)
- \( N \): Endogenous tax. This label must be followed by the name of an auxiliary variable.
- \( M \): Endogenous tax multiplier. The advalorem tax rate is the product of the value of the endogenous tax and this multiplier.

\( a;,b;,… \) Nesting assignments. Only one such label may appear per line.

The valid labels in an \( E; \) line include:

- \( Q \): Reference quantity
- \( R \): Rationing instrument indicating an auxiliary variable.

The valid labels in a \( D; \) line include:

- \( Q \): Reference quantity
- \( P \): Reference price

\( a;,b;,… \) Nesting assignment

**Constraints**

Auxiliary constraints in MPSGE models conform to standard \( GAMS \) equation syntax. The may refer to any of the four classes of variables, \( \$SECTORS, \$COMMODITIES, \$CONSUMERS \) and \( \$AUXILIARY \), but they may not reference variables names declared within a \( \$REPORT \) block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated POSITIVE (the default), the auxiliary constraint should be expressed as a "greater or equal" inequality (\( =G= \)). If an auxiliary variable is designated FREE, the associated constraint must be expressed as an equality (\( =E= \)).

\[
\$CONSTRAINT:TAU
\begin{align*}
G & =G= X \ast Y; \\
\$CONSTRAINT:MU(I)$MU0(I)
\end{align*}
\]

\[
MU(I) \ast P(I) \ast Q(I) =G= \text{SUM}(J, \text{THETA}(I,J) \ast PX(J));
\]
The exception applied in this example restricts the equation only to those elements of set $I$ for which $MU_0(I)$ is not zero.

**Report Declaration**

The GAMS interface to MPSGE normally returns level values only for the central variables - those declared within $\$SECTORS$, $\$COMMODITIES$, $\$CONSUMERS$: and $\$ AUXILIARY$: sections. An equilibrium determines not only these values, but also levels of demand and supply by individual sectors and consumers. Given benchmark information, elasticities and the equilibrium values, all such demands can be computed, but this can be tedious to do by hand. In order to have these values returned by MPSGE, it is necessary to indicate the name of the variable into which the value is to be returned. The general form is as follows:

\[
\begin{align*}
\$REPORT: & \\
V: & \text{variable name} & I: & \text{commodity} & \text{PROD:} & \text{sector} \\
V: & \text{variable name} & O: & \text{commodity} & \text{PROD:} & \text{sector} \\
V: & \text{variable name} & D: & \text{commodity} & \text{DEMAND:} & \text{consumer} \\
V: & \text{variable name} & W: & \text{consumer} & \text{} & \text{} \\
\end{align*}
\]

The first row returns an input quantity, the second row returns an output quantity, the third returns a demand quantity, and the fourth row returns a consumer welfare index. (Note: the level value returned for a "consumer variable" is an income level, not a welfare index.)

\[
\begin{align*}
\$REPORT: & \\
V: & DL(S) & I: & \text{PF("L")} & \text{PROD:} & Y(S) \\
V: & DK(S) & I: & \text{PF("K")} & \text{PROD:} & Y(S) \\
V: & SX(G,S) & D: & PX(G) & \text{PROD:} & X(S) \\
V: & D(G,H) & D: & P(G) & \text{DEMAND:} & RA(H) \\
V: & W(H) & W: & RA(H) & \text{} & \text{} \\
\end{align*}
\]

Note that the "$" exception is only meaningful on the first entry. Also notice that the domain of the report variable must conform to the domain of the subsequent two entries.

**Differences between Scalar and Vector MPSGE Syntax**

1. $\$MODEL$: statement The $\$MODEL$ statement is required in the vector format and it must precede all other statements. A name which is an acceptable file name prefix must be used. The preprocessor does not begin translation of an MPSGE model until it encounters a $\$MODEL$ statement following an $\$ONTEXT$ record. The preprocessor continues to translate until it reaches an $\$OFFTEXT$ statement, skipping blank lines and comment lines identified by a "∗" in column 1.

2. Case folding: In the vector syntax, upper and lower case letters are not distinguished. The entire file is processed as though it were written in upper case. This is not compatible with the earlier version of MPSGE in which "P" and "p" were distinct.

3. Distinct names: Names used for variables in the MPSGE model must be distinct from each other as well as from all other symbols in the GAMS program. If there is a GAMS set or parameter or model named $X$, then $X$ may not be used to identify an MPSGE sector or commodity.

4. Tabs: MPSGE fields are free format and tabs are translated to spaces by the preprocessor. Tabs are permitted in GAMS provided that the compiler is properly configured (under DOS, "TABIN 8" must be inserted in file GAMSPARM.TXT in the GAMS system directory).

**Exception Handling**

The GAMS exception operator can be used on virtually any entry in the MPSGE input file. For example, if you want to have sector $X(S)$ have one production structure for elements $S$ in a subset $T(S)$, you can provide separate production function declarations as follows:
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1329

\$
\text{PROD:} X(S) \text{T}(S) \\
\ldots \quad \text{! sector X described for S in T} \\
\$

\$
\text{PROD:} X(S) \text{(NOT T(S))} \\
\ldots \quad \text{! sector X described for S not in T.} \\
\$

The preprocessor does not require one and exactly one declaration for each sector. If multiple declarations appear, the later set of coefficients overwrites the earlier set.

Switches and Debug Output

Run-time tolerances and output switches may be specified within the vector-syntex model specification or using the PUT facility, they can be written directly to the MPS input file. Output switches control the level of debug output written by the MPSGE subsystem to the solver status file. Reports provided by \$ECHOP, \$FUNLOG and \$DATECH can be returned to the listing file by specifying "OPTION SYSOUT=ON;" within the GAMS program prior to the SOLVE statement. The recognized MPSGE parameters are:

\$
\text{ECHOP: logical Default=.FALSE.} \\
\$

is a switch for returning all or part of the scalar MPSGE file to the solver status file. In order to have this output printed in the listing file, enter the GAMS statement "OPTION SYSOUT=ON;" prior to solving the model.

\$
\text{PEPS: real Default=1.0E-6} \\
\$

is the smallest price for which price-responsive demand and supply functions are evaluated. If a price is below PEPS, it is perturbed (set equal to PEPS) prior to the evaluation.

\$
\text{EULCHK: logical Default=.TRUE.} \\
\$

is a switch for evaluating Euler's identity for homogeneous equations. The output is useful for monitoring the numerical precision of a Jacobian evaluation. When a price or income level is perturbed in a function, the Euler check may fail.

\$
\text{WALCHK: logical Default=.TRUE.} \\
\$

is a switch for checking Walras's law. Like EULCHK, this switch is provided primarily to monitor numerical precision. When an income level is perturbed, the Walras check may fail.

\$
\text{FUNLOG: logical Default=.FALSE.} \\
\$

is a switch to generate a detailed listing of function evaluations for all production sectors and consumers.

FUNLOG triggers a function evaluation report which provides detailed output describing the evaluation of supply and demand coefficients. The information provide is sufficient that an industrious graduate student should be able to reproduce the results (given a pencil, paper and slide rule).

The evaluation report has the following headings:
When `$FUNLOG:.TRUE.` is specified, a complete report of demand and supply coefficients for every production and demand function in every iteration. Be warned that with large models this can produce an enormous amount of output!

The following two function evaluation reports are generated in the first iteration in solving case "L" for model `HARBERGER`:

Function Evaluation for: `AL.X`

<table>
<thead>
<tr>
<th>T</th>
<th>N</th>
<th>PBAR</th>
<th>P</th>
<th>QBAR</th>
<th>Q</th>
<th>KP</th>
<th>ELAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
<td>s</td>
<td>1.0000E+00</td>
<td>8.9198E-01</td>
<td>1.0000E+02</td>
<td>1.0000E+02</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>OA</td>
<td>t</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>1.0000E+02</td>
<td>1.0000E+02</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>IA</td>
<td>a</td>
<td>1.0000E+00</td>
<td>8.7998E-01</td>
<td>9.0000E+01</td>
<td>9.0000E+01</td>
<td>s</td>
<td>1.00</td>
</tr>
<tr>
<td>D</td>
<td>P.X</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>1.0000E+02</td>
<td>1.0000E+02</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>P.Y</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>1.0000E+01</td>
<td>1.0000E+01</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>W.K</td>
<td>2.0000E+00</td>
<td>1.5000E+00</td>
<td>2.0000E+01</td>
<td>2.3466E+01</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>W.L</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>5.0000E+01</td>
<td>4.3999E+01</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

Function evaluation for: `RA.OWNER`

<table>
<thead>
<tr>
<th>T</th>
<th>N</th>
<th>PBAR</th>
<th>P</th>
<th>QBAR</th>
<th>Q</th>
<th>KP</th>
<th>ELAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
<td>s</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>7.0000E+01</td>
<td>7.0000E+01</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>OA</td>
<td>t</td>
<td>1.0000E+00</td>
<td>0.0000E+00</td>
<td>0.0000E+00</td>
<td>0.0000E+00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>IA</td>
<td>a</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>7.0000E+01</td>
<td>7.0000E+01</td>
<td>s</td>
<td>0.50</td>
</tr>
<tr>
<td>D</td>
<td>P.X</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>3.0000E+01</td>
<td>3.0000E+01</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>P.Y</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>4.0000E+01</td>
<td>4.0000E+01</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>W.K</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>6.0000E+01</td>
<td>0.0000E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>PT</td>
<td>1.0000E+00</td>
<td>1.0000E+00</td>
<td>1.0000E+01</td>
<td>0.0000E+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`$DATECH`: logical Default=.FALSE.

is a switch to generate a annotated listing of the function and Jacobian evaluation including a complete listing of all the nonzero coefficients.
MPSGE generates an analytic full first-order Taylor series expansion of the nonlinear equations in every iteration. Nonzero elements of the Jacobian matrix are passed to the system solver (MILES or PATH) which uses this information in the direction-finding step of the Newton algorithm. Coefficients are produced with codes which help interpret where they came from. The following codes are used:

- **W0** indicates an element from the orthogonal part of \( F() \).
- **W1** indicates an element from the non-orthogonal part of \( F() \).
- **B** indicates a linear term from \( F \).
- **E0** indicates a homogeneous Jacobian entry.
- **E1** indicates a non-homogeneous Jacobian entry.

The Euler checksum examines elements from the linearization which are type "E0". The Walras checksum examines elements from the function evaluation which are type "W0".

Needless to say, the \$DATECH: .TRUE. switch produces a very big status file for large models. It is not something which is very useful for the casual user.

Here is a partial listing of nonzeros generated during the first linearization for scenario "L" in model HARBERGER:

```
---------------------------------------- Coefficients for sector:AL.X
  P.X  AL.X       1.0000E+02  B
  AL.X  P.X       -1.0000E+02  B
  P.Y  AL.X       -1.0000E+01  B
  AL.X  P.Y       1.0000E+01  B
  W.K  AL.X       -2.3466E+01  B
  AL.X  W.K       3.5199E+01  B
  W.L  AL.X       -4.3999E+01  B
  AL.X  W.L       4.3999E+01  B
  W.K  W.K       1.3037E+01  E0  1.3037E+01
  W.K  W.L       -1.3037E+01  E0 -1.3037E+01
  W.L  W.K       -1.9555E+01  E0 -1.9555E+01
  W.L  W.L       1.9555E+01  E0  1.9555E+01
  GOVT  AL.X      -1.1733E+01  B
  GOVT  W.K      -1.1733E+01  E1
  GOVT  W.K      6.5184E+00  E0  6.5184E+00
  GOVT  W.L      -6.5184E+00  E0 -6.5184E+00

---------------------------------------- Income for consumer:RA.OWNER
  W.K                  6.0000E+01  W0
  RA.OWNER  W.K      -6.0000E+01  B
  PT                  1.0000E+01  W0
  RA.OWNER  PT      -1.0000E+01  B
  RA.OWNER  RA.OWNER  1.0000E+00  B

---------------------------------------- Demands for consumer:RA.OWNER
  P.X                -3.0000E+01  W0
  P.Y                -4.0000E+01  W0
  P.X  P.X         8.5714E+00  E0  8.5714E+00
  P.X  P.X         1.2857E+01  E0  1.2857E+01
  P.X  P.Y         -8.5714E+00  E0 -8.5714E+00
  P.X  P.Y         1.7143E+01  E0  1.7143E+01
  P.Y  P.X         -8.5714E+00  E0 -8.5714E+00
  P.Y  P.X         1.7143E+01  E0  1.7143E+01
  P.Y  P.Y         8.5714E+00  E0  8.5714E+00
  P.Y  P.Y         2.2857E+01  E0  2.2857E+01
  P.X  RA.OWNER    -4.2857E-01  E0 -3.0000E+01
  P.Y  RA.OWNER    -5.7143E-01  E0 -4.0000E+01
```
4.53.2.8 Appendix B: File Structure

This appendix provides an overview of the structure of GAMS input files which include MPSGE models. The text of the paper presents many of these ideas by way of example, but it may also be helpful for some users to have a "template" for constructing MPSGE models. The discussion in this section focuses on a "generic" input file, the schematic form of which is presented in Table 10. This section first presents a "top down" view of program organization, and then it discusses aspects of the new syntax for model specification.

Flow of Control

When a model is developed using GAMS as a front-end to MPSGE, the input file generally has five sections as identified in Table 10. Section (i), the benchmarking section, contains standard GAMS statements. This includes GAMS SET declarations, input data (SCALARS, PARAMETERS and TABLES), and PARAMETER declarations for intermediate arrays used in benchmarking or model specification. In complex models, this section of the file will typically contain some algebraic derivations, the result of which is a calibrated benchmark equilibrium dataset.

Users who are unfamiliar with GAMS can consult the manual. Beginning GAMS programmers should remember that the MPSGE interface to GAMS is unlike other solution subsystems. "Level values" are passed between the GAMS program and MPSGE in the usual fashion, but MPSGE models do not require the explicit use of the VARIABLE or EQUATION statements.)

The second section of the file consists of a GAMS comment range, beginning with an $ONTEXT record and ending with an $OFFTEXT record, followed by an invocation of the preprocessor. The preprocessor writes operates on statements in the MPSGE model declaration which are "invisible" to the GAMS compiler. This program reads the MPSGE model statements and generates GAMS-readable code, including a model_name.gen file. Additional GAMS code produced by the preprocessor includes declarations for each of the central variables and report variables in the MPSGE model.

The third section of the generic input file performs a "benchmark replication" and may not be present in all applications. There are four statements required for benchmark validation. The first statement sets the iteration limit to be zero; the second statement causes the MPSGE model to be "generated", and the third statement causes the MPSGE solver to read the model and return the deviations. In this call, the level values passed to the solver are unaltered because the iteration limit is zero. Market excess supplies and zero profit checks are returned in the "marginals" of the associated commodity prices and activity levels, respectively. The final statement in this section resets the iteration limit to 1000 (the default value) for subsequent counter-factual computations.

Section (iv) defines and then computes a counter-factual equilibrium. A counter-factual equilibrium is defined by parameter values such as tax rates or endowments which take on values different from those in the benchmark equilibrium. Within the GAMS interface to MPSGE, it is also possible to fix one or more central variables. When any variable is fixed, the associated equation is omitted from the equilibrium system during the solution process but the resulting imbalance is then reported in the solution returned through the marginal.

The final section of the file represents the GAMS algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

The final section of the file represents the GAMS algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

For large models, the advantage of the vector format is that by using appropriately defined GAMS sets, the number of individual functions which need to be defined is reduced only to the number of "classes" of functions. This makes it possible to represent large dimensional models using only a few lines of code.

To summarize, here are the basic features of a program which uses GAMS as a front-end to MPSGE:
1. An MPSGE model is defined within a GAMS comment range followed by
   
   $\text{sysinclude mpsgeset model_name}$

2. Every SOLVE statement for a particular model is preceded by $\text{INCLUDE MODEL.GEN}$. The GEN file is written by the preprocessor based on the model structure.

3. Solution values for the central variables in the MPSGE model and any declared "report variables" are returned in GAMS variable level values. Level values for slacks are returned as "marginals" for the associated variables.

4. The model description follows a format which is a direct extension of the scalar data format. Certain aspects of the new language, such as case folding, are incompatible with the original MPSGE syntax.

GAMS Code Generated by the Preprocessor: the GEN File

Most novice users will find it easiest to treat the preprocessor output files as "black boxes". These files contain GAMS source code required for declaring and generating the MPSGE input file. Table 11 contains portions of the GEN file for the same model. Table 12 shows the preprocessor-generated listing and symbol table which are always appended to the bottom of the GEN file. If a preprocessor error occurs, it can be helpful to consult the symbol table to track down the bug. Finally, Table 13 shows the first page of scalar format MPSGE input file produced by HARBERGER.GEN. Normally, this file is written and then erased in the course of a GAMS run, although all or part of the file may be retained using the $\text{ECHOP}$: switch.

4.53.3 Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE

This research supported by the GAMS Applied General Equilibrium Research Fund. The software described here operates only with GAMS 2.25.085 or later on the PC, shipped in July, 1995. The author remains responsible for any bugs which exist in this software.

Author

Thomas F. Rutherford, rutherford@colorado.edu Department of Economics University of Colorado

Date

1995

4.53.3.1 An Overview

This document describes a mathematical programming system for general equilibrium analysis named MPSGE which operates as a subsystem to the mathematical programming language GAMS. MPSGE is library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. MPSGE simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE separates the tasks of model formulation and model solution, thereby freeing model builders from the tedious task of writing model-specific function evaluation subroutines. All features of a particular
model are communicated to GAMS/MPSGE through a tabular input format. To use MPSGE, a user must learn the syntax and conventions of this model definition language.

The present paper is intended for students who have completed two semesters of study in microeconomics. The purpose of this presentation is to give students a practical perspective on microeconomic theory. The diligent student who works through all of the examples provided here should be capable of building small models "from scratch" to illustrate basic theory. This is a first step to acquiring a set of useable tools for applied work.

The remainder of this paper is organized as follows. Section The Theory of Consumer Demand review the theory of consumer demand; section Getting Started provides guidance on the mechanics of numerical modeling, including instructions on how to install and test the GAMS/MPSGE software; section Modeling Consumer Demand introduces the modeling framework with three models illustrating the representation of consumer demand within the MPSGE language. Section The Pure Exchange Model reviews the pure exchange model, and section Modeling Pure Exchange with MPSGE presents two MPSGE models of exchange. Each of the model-oriented sections present exercises based on the models which give students a chance to work through the material on their own. Additional introductory examples for self-study can be found in the Markusen library as well as in the GAMS model library (look for models with names ending in "MGE").


The ultimate objective of this piece is to remind students of some theory which they have already seen and illustrate how these ideas can be used to build numerical models using GAMS with MPSGE. It is not my intention to provide a graduate level presentation of this material. So far as possible, I have avoided calculus and even algebra. The objective here is to demonstrate that what matters are economic ideas. With the proper tools, it is possible to do concrete economic modeling without a lot of mathematical formalism.

### 4.53.3.2 The Theory of Consumer Demand

A central idea underlying most microeconomic theory is that agents optimize subject to constraints. The optimizing principle applied to consumer choice begins from the notion that agents have preferences over consumption bundles and will always choose the most preferred bundle subject to applicable constraints. To operationalize this theory, three issues which must be addressed:

1. How can we represent preferences?
2. What is the nature of constraints on consumer choice? and
3. How can the choice be modelled?

Preferences are relationships between alternative consumption "bundles". These can be represented graphically using "indifference curves", as illustrated in Figure 1. Focusing now on the preferences of a single consumer, the indifference curve is a line which connects all combinations of two goods x and y between which our consumer is indifferent. As this curve is drawn, we have represented an agent with "well-behaved preferences": at any allocation, more is better (monotonicity), and averages are preferred to extremes (convexity). Exactly one such indifference curve goes through each positive combination of x and y. Higher indifference curves lie to the "north-east".

Figure 1: An Indifference Curve
If we wish to characterize an agent's preferences, the "marginal rate of substitution" (MRS) is a useful point of reference. At a given combination of $x$ and $y$, the marginal rate of substitution is the slope of the associated indifference curve. As drawn, the MRS increases in magnitude as we move to the northwest and the MRS decreases as we move to the southeast. The intuitive understanding is that the MRS measures the willingness of the consumer to trade off one good for the other. As the consumer has greater amounts of $x$, she will be willing to trade more units of $x$ for each additional unit of $y$ – this results from convexity.

An "ordinal" utility function $U(x, y)$ provides a helpful tool for representing preferences. This is a function which associates a number with each indifference curve. These numbers increase as we move to the northeast, with each successive indifference curve representing bundles which are preferred over the last. The particular number assigned to an indifference curve has no intrinsic meaning. All we know is that if $U(x_1, y_1) > U(x_2, y_2)$, then the consumer prefers bundle 1 to bundle 2.

Figure 2 illustrates how it is possible to use a utility function to generate a diagram with the associated indifference curves. This figure illustrates Cobb-Douglas well-behaved preferences which are commonly employed in applied work.

Figure 2: Indifference Curves and Utility Levels

Up to this point, we have we have focused exclusively on the characterization of preferences. Let us now consider the other side of the consumer model – the budget constraint. The simplest approach to
characterizing consumer income is to assume that the consumer has a fixed money income which she may spend on any goods. The only constraint on this choice is that the value of the expenditure may not exceed the money income. This is the standard budget constraint:

\[ Px \, x + Py \, y = M. \]

Graphically, this equation defines the line depicted in Figure 3. All points inside the budget line are affordable. The consumer faces a choice of which affordable bundle to select.

**Figure 3: The Budget Set**

Within the framework of our theory, the answer is straightforward – the consumer will choose the one combination of x and y from the set of affordable bundles which maximizes her utility. This combination of x and y will be at the point where the indifference curve is tangent the budget line. This point is called the optimal choice. We see this illustrated in Figure 4.
Figure 4: Utility Maximization on the Budget Set

The standard model of consumer behavior provides a starting point for learning MPSGE. This introduction is "hands on" – I will discuss issues as they arise, assuming that you have access to a computer and can invoke the program and read the output file. You may wish to learn the rudiments of GAMS syntax before starting out, although you may be able to pick up these ideas as they come, depending on your aptitude for computer languages.

4.53.3 Getting Started

1. You need to know how to generate a standard text file. There are several methods for doing this. One approach is to use EDIT, the standard DOS text editor. Another approach favored by many people who have used text processors like WordPerfect or Microsoft Word is to use the word processor as a text editor. A disadvantage of this approach is that you will need to remember to always save the edited file in a text format. A final approach, one which I suggest to graduate students who are interested in numerical modeling, is that you develop facility with a "real" programmer's editor like Emacs, Epsilon or Brief. These editors are far more powerful than the DOS Edit, and they are far better suited to computational work than a word processor.

2. You need to have a copy of GAMS with MPSGE to run on your computer. There are versions of this program for 386/486/586 PCs as well as for most widely-used workstations (SUN, DEC, HP, IBM, etc.). Copies of the program can be obtained directly from GAMS (gams@gams.com), or you may take a copy from someone who already has the program. (Copyright restrictions apply only to GAMS license files.) You may copy the GAMS programs without the license files and the program will only operate in student/demonstration mode. The student version is perfectly adequate for learning about modeling – in fact, it may be better because it's dimensionality restrictions prevent the novice model builder from adding unnecessary details.

3. You need to install GAMS with MPSGE on your computer. To do this, follow the standard installation procedure:

   (a) Make a GAMS system directory, e.g.
   c:\>mkdir gams

   (b) Copy all the files from the GAMS distribution diskette(s) into the GAMS system directory; e.g.
   c:\>copy a:\.*.* c:\gams

   (c) Connect to the GAMS system directory and run the GAMS installation program e.g.
   c:\>cd gams
c:\>gams\gamsinst

   The installation program gives you choices regarding which solver you wish to use for various problem classes. For MPSGE models, the key choice is the MCP solver. I recommend that you select MPSGE if you have this solver on your computer. It is the most efficient and robust solver currently available for this class of models. (MPSGE by Michael Ferris and Steve Dirkse of the University of Wisconsin at Madison.)

   You complete the installation by adding the GAMS system directory to your DOS environment variable, MPSGE. Normally, MPSGE is initialized at startup in the file C:\AUTOEXEC.BAT. Look for a line which looks like "PATH C:\DOS" and modify this to read something like: "PATH C:\DOS;C:\GAMS" using a text editor.

4. You should verify that the system is operational. Connect to a working directory (never run models from the GAMS system directory!). Then, extract and run one of the library models, e.g.
If the GAMS system is properly installed, these commands will cause GAMS to solve a sequence of models from the SCARFMGE sample problem. The output from this process is written to file SCARFMGE.LST. If you search for the word "STATUS", you can verify that all the cases are processed.

There are a number of MPSGE models included in the GAMS library. If you are using a student version of GAMS, you will be able to process some but not all of the library models. The student version of the program limits the number of variables in the model to 100. (I believe that GAMS imposes other limits on use of the student version, but the variable limitation is the most severe constraint.)

Assuming that you have successfully installed the software, let us now proceed to some examples which illustrate both the computing syntax and the underlying economics.

4.53.3.4 Modeling Consumer Demand

4.53.3.4.1 Example 1: Evaluating a Demand Function

Consider a standard consumer choice problem, one which might appear on a midterm examination in intermediate microeconomics:

\[
\begin{align*}
\text{max} & \quad U(x, y) = \ln(x) + 2 \ln(y) \\
\text{subject to:} & \quad lx + 2y = 120
\end{align*}
\]

where 1 is the exogenous price of x and 2 is the price of y.

This type of problem is solved easily using GAMS/MINOS (as a nonlinear program). Strictly speaking, it is not the sort of model for which you would need to use MPSGE. At the same time, this can be an instructive example.

The key issue in this example is learning how to represent utility functions. MPSGE is "non-algebraic"—so function specification depends on an intuitive understanding of the underlying economic structure.

Consider Figure 5 and focus on a single point, \(x = 1, y = 1\). There is an indifference curve through this point, and the marginal rate of substitution (MRS) at this point is simply the slope of this curve. The benchmark MRS does not uniquely determine the underlying preferences.

**Figure 5: A Calibrated Benchmark**

A utility function is represented in MPSGE by the specification of: (i) benchmark demand quantities, (ii) benchmark demand prices (iii) an elasticity of substitution at the benchmark point. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point, and the elasticity describes the curvature of the indifference curve. Speaking formally, elasticities provide a "second order approximation" of the utility function. To understand the importance of the benchmark elasticity of substitution, consider Figure 6. This figure shows three indifference curves all of which share the same benchmark quantities and benchmark prices. They differ only in the elasticities of substitution. The least convex (flattest) curve has the highest elasticity, the most convex curve has the lowest elasticity. (When the elasticity of substitution is 0, the indifference curve is L-shaped with the corner at the benchmark point.)
Let us now consider how the consumer optimization problem can be cast as a general equilibrium model. We do this by adding a single factor of production and two "production" sectors. For concreteness, let the factor of production be called labor with a price PL. One production function converts one unit of labor into one unit of $x$, the other sector converts 2 units of labor into one unit of $y$. Setting the labor endowment equal 120, the market clearance condition for labor reads:

\[ x + 2y = 120 \]

which is precisely the budget constraint for the consumer's problem.

We will now present the program code, a few lines at a time. As part of working through the example, the student should type these lines into a file.

A MPSGE model specification is always listed between $\text{ONTEXT}$ and $\text{OFFTEXT}$ statements. The first statement within an MPSGE model-description assigns a name to the model. The model name must begin with a letter and must have 10 or fewer characters.

$\text{ONTEXT}$

$\text{MODEL:DEMAND}$
The model specification begins by declaring variables for the model. In a standard model, there are three types of variables: commodity prices, sectoral activity levels, and consumer incomes. The end of each line may include "! variable description ".

_N.B. The variables associated with commodities are prices, not quantities. (In this and subsequent models, I use P as the first letter for each of the commodity variables to remind us that these variables are prices.)_

_N.B. The variable associated with a consumer is an income level, not a welfare index._

```plaintext
$SECTORS:
  X ! ACTIVITY LEVEL FOR X = DEMAND FOR GOOD X
  Y ! ACTIVITY LEVEL FOR Y = DEMAND FOR GOOD Y

$COMMODITIES:
  PX ! PRICE OF X WHICH WILL EQUAL PL
  PY ! PRICE OF Y WHICH WILL EQUAL 2 PL
  PL ! PRICE OF THE ARTIFICIAL FACTOR L

$CONSUMERS:
  RA ! REPRESENTATIVE AGENT INCOME
```

Function specifications follow the variable declarations. In this model, our first declarations correspond to the two production sectors. In this model, the production structures are particularly simple. Each of the sectors has one input and one output. In the MPSGE syntax, _I_: denotes an input and _O_: denotes an output. The output quantity coefficients for both sectors are unity (_Q:1_). This means that the level values for x and y correspond to the actual quantities produced. The final function specified in the model represents the utility function and endowments for our single consumer. In this function, the _E_: entries correspond to endowments and the _D_: entries are demands. Reference demands, reference prices and the substitution elasticity (_s:1_) characterize preferences.

The demand entries shown here are consistent with a Cobb-Douglas utility function in which the budget share for y is twice the budget share for x (i.e. the MRS at (1,1) equals 1/2):

```plaintext
$PROD:X
  O:PX Q:1
  I:PL Q:1

$PROD:Y
  O:PY Q:1
  I:PL Q:2

$DEMAND:RA s:1
  E:PL Q:120
  D:PX Q:1 P:(1/2)
  D:PY Q:1 P:1
```

The final three statements in this file invoke the MPSGE preprocessor, ”generate” and solve the model:

```plaintext
$SYSINCLUDE mpsgeset DEMAND

$INCLUDE DEMAND.GEN

SOLVE DEMAND USING MCP;
```
The preprocessor invocation ($SYSINCLUDE mpsget$) should be placed immediately following the $OFFTEXT block containing the model description. The model generator code, DEMAND.GEN, is produced by the previous statement and must be referenced immediately before each subsequent SOLVE statement.

At this point, the reader should take the time to type the example into a file and execute the program with GAMS/MPSGE.

This is possibly the first GAMS model which some readers have solved, so it is worth looking through the listing file in some detail. After running the solver, we examine the listing file. I typically begin my assessment of a model's solution by searching for "STATUS". For this model, we have the following:

```
SOLVE SUMMARY

MODEL DEMAND
TYPE MCP
SOLVER PATH FROM LINE 263

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 1 OPTIMAL

RESOURCE USAGE, LIMIT 1.432 1000.000
ITERATION COUNT, LIMIT 5 1000
EVALUATION ERRORS 0 0

Work space allocated -- 4.86 Mb
```

Default price normalization using income for RA

This information is largely self-explanatory. The most important items are the SOLVER STATUS and MODEL STATUS indicators. When the solver status is 1 and the model status is 1, the system has returned an equilibrium.

For small models such as this, the limits on resource usage (time) and solver iterations have no effect. (You can modify these values with the statements:

```
model.RESLIM = number of cpu seconds ;
model.ITERLIM = number of iterations ;
```

entered into the program before the SOLVE statement.)

The work space allocation for MPSGE models is determined by the number of variables in the model. It is possible to exogenously specify the work space allocation by assigning

```
model.WORKSPACE = xx ;
```

where xx is the desired number of megabytes.

The final message, "Default price normalization...", is significant. It reminds the user that an Arrow-Debreu general equilibrium model determines only relative prices. In such an equilibrium, the absolute scaling of prices is indeterminant. (I.e., if $(p^*, M^*)$ are a set of equilibrium prices and income levels, then $(2 p^*, 2 M^*)$ is also a solution, etc.)

It is common practice in economics to address the normalization issue through the specification of a numeraire commodity. You can do this for an MPSGE model by "fixing" a price, with a statement like:
entered following the model declaration ($SYSINCLUDE mpsgeset) but prior to the solver invocation. When any price or income level is fixed, MPSGE recognizes that a numeraire has been specified and does no automatic normalization.

Following some output from the solver (PATH in this case), the listing file provides a complete report of equilibrium values. With MPSGE models, the equation listings are superfluous. The variable listings provide all the relevant information.

For this model, the solution listing appears as follows:

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>VAR X</td>
<td>40.000</td>
<td>+INF</td>
</tr>
<tr>
<td>----</td>
<td>VAR Y</td>
<td>40.000</td>
<td>+INF</td>
</tr>
<tr>
<td>----</td>
<td>VAR PX</td>
<td>1.000</td>
<td>+INF</td>
</tr>
<tr>
<td>----</td>
<td>VAR PY</td>
<td>2.000</td>
<td>+INF</td>
</tr>
<tr>
<td>----</td>
<td>VAR PL</td>
<td>1.000</td>
<td>+INF</td>
</tr>
<tr>
<td>----</td>
<td>VAR RA</td>
<td>120.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

X SUPPLY AND DEMAND OF GOOD X
Y SUPPLY AND DEMAND OF GOOD Y
PX PRICE OF X WHICH WILL EQUAL CX * PL
PY PRICE OF Y WHICH WILL EQUAL CY * PL
PL PRICE OF THE ARTIFICIAL FACTOR L
RA REPRESENTATIVE AGENT INCOME

The LOWER and UPPER columns report variable bounds applied in the model. In these columns, zero is represented by "." and infinity is represented by "+INF". The LEVEL column reports the solution value returned by the algorithm. Here we see that the equilibrium price of $x$ is 1 and the price of $y$ is 2, as determined by the specification of labor inputs.

4.53.3.4.2 Exercises 1 (a) The utility function calibration point is arbitrary. Here, we have selected $x = y = 1$ as the reference quantity. Revise the program to use a different calibration point where $x = 2$ and $y = 1$, where $MRS(2, 1) = 1/4$. (Remember to modify both the $Q:$ and $P:$ fields.) Rerun the model to demonstrate that this does not change the result.

(b) Increase the price of $x$ from 1 to 2 by changing the $Q:$ coefficient for PL in sector X from 1 to 2. What happens to the demand for $x$? Explain why a change in the price of $x$ is represented by a change in the $Q:$ field for sector X.

(c) Compute an equilibrium in which commodity $y$ is defined as the numeraire.

4.53.3.4.3 Example 2: Evaluating the MRS This example further explores the representation of demand functions with MPSGE. It sets up a trivial equilibrium model with two goods and one consumer which returns the marginal rate of substitution of good $x$ for good $y$ at a given level of demand. The underlying utility function is:

$$U(x, y) = \ln(x) + 4\ln(y)$$

When $x = y = 1$, the marginal rate of substitution of $x$ for $y$ is $1/4$. We use this information to calibrate the demand function, specifying the ratio of the reference prices of $x$ to $y$ equal to $1/4$. 
In an equilibrium, final demand always equals endowments for both goods, because these are the only sources of demand and supply. The model as parameterized demonstrates that if we set endowments for this model equal to the demand function calibration point, the model equilibrium price ratio equals the benchmark MRS.

This program begins with some GAMS statements in which three scalar parameters are declared. These parameters will be used in the place of numbers within the MPSGE model. The syntax for these GAMS statements is introduced in Chapter 2 of the GAMS manual. In short, we declare $x$, $y$ and MRS as scalar parameters and initialize the first two of these to unity. The MRS parameter is assigned a value following the solution of the model.

SCALAR
  X  QUANTITY OF X FOR WHICH THE MRS IS TO BE EVALUATED /1/
  Y  QUANTITY OF Y FOR WHICH THE MRS IS TO BE EVALUATED /1/
  MRS COMPUTED MARGINAL RATE OF SUBSTITUTION;

The remainder of the MPSGE program is, in fact, simpler than Example 1.

$ONTEXT
$MODEL:MRSCAL

$COMMODITIES:
  PX   ! PRICE INDEX FOR GOOD X
  PY   ! PRICE INDEX FOR GOOD Y

$CONSUMERS:
  RA   ! REPRESENTATIVE AGENT

$DEMAND:RA  s:1
  D:PX  Q:1  P:(1/4)
  D:PY  Q:1  P:1
  E:PX  Q:X
  E:PY  Q:Y

$OFFTEXT
$SYSINCLUDE mpsgeset MRSCAL

$INCLUDE MRSCAL.GEN
SOLVE MRSCAL USING MCP;

Following the solution, we compute a function of the solution values, the ratio of the price of $x$ to the price of $y$. We do this using the GAMS syntax which references the equilibrium level values of the PX and PY and storing this result in the scalar MRS. This scalar value is then displayed in the listing file with 8 digits:

\[ \text{MRS} = \frac{\text{PX.L}}{\text{PY.L}}; \]

OPTION MRS:8;
DISPLAY MRS;

4.53.3.4.4 Exercises 2 (a) Show that the demand function is homothetic by uniform scaling of the $x$ and $y$ endowments. The resulting MRS should remain unchanged.

(b) Modify the demand function calibration point so that the reference prices of both $x$ and $y$ equal unity (hint: the marginal rate of substitution is:

\[ \text{MRS} = \frac{x}{4y}. \]
4.53.3.4.5 Example 3: Leisure Demand and Labor Supply  This model investigates the labor-leisure decision. A single consumer is endowed with labor which is either supplied to the market or "repurchased" as leisure. The consumer utility function over market goods (x and y) and leisure is Cobb-Douglas:

\[ U(x, y, L) = \ln(x) + \ln(y) + \ln(L) \]

Goods x and y may only be purchased using funds obtained from labor sales. This constraint is written:

\[ x + y = \text{LPROD} \times \text{LS} \]

where goods x and y both have a price of unity at base year productivity and LPROD is an index of labor productivity. An increase in productivity is equivalent to a proportional decrease in the prices of x and y.

The model declaration is as follows:

```plaintext
SCALAR LPROD AGGREGATE LABOR PRODUCTIVITY /1/
       CX COST OF X AT BASE YEAR PRODUCTIVITY /1/
       CY COST OF Y AT BASE YEAR PRODUCTIVITY /1/;
$ONTEXT
$MODEL:LSUPPLY
$SECTORS:
   X ! SUPPLY=DEMAND FOR X
   Y ! SUPPLY=DEMAND FOR Y
   LS ! LABOR SUPPLY
$COMMODITIES:
   PX ! MARKET PRICE OF GOOD X
   PY ! MARKET PRICE OF GOOD Y
   PL ! MARKET WAGE
   PLS ! CONSUMER VALUE OF LEISURE
$CONSUMERS:
   RA ! REPRESENTATIVE AGENT
$PROD:LS
   O:PL Q:LPROD
   I:PLS Q:1
$PROD:X
   O:PX Q:1
   I:PL Q:CX
$PROD:Y
   O:PY Q:1
   I:PL Q:CY
$DEMAND:RA s:1
   E:PLS Q:120
   D:PLS Q:1   P:1
   D:PX Q:1   P:1
   D:PY Q:1   P:1
$OFFTEXT
```
We can use this model to evaluate the wage elasticity of labor supply. In the initial equilibrium (computed in the last statement) the demands for $x$, $y$ and $L$ all equal 40. A subsequent assignment to $LPROD$ (below) increases labor productivity. After computing a new equilibrium, we can use the change in labor supply to determine the wage elasticity of labor supply, an important parameter in labor market studies. It should be emphasized that the elasticity of labor supply should be an input rather than an output of a general equilibrium model – this is a parameter for which econometric estimates can be obtained.

Here is how the programming works. First, we declare some scalar parameters which we will use for reporting, then save the "benchmark" labor supply in $LS0$:

```
SCALAR
  LS0   REFERENCE LEVEL OF LABOR SUPPLY
  ELS   ELASTICITY OF LABOR SUPPLY WRT REAL WAGE;

LS0 = LS.L;
```

Next, we modify the value of scalar $LPROD$, increasing labor productivity by 1%. Because this is a neoclassical model, this change is equivalent to increasing the real wage by 1%. We need to recompute equilibrium prices after having changed the $LPROD$ value:

```
LPROD = 1.01;
$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;
```

We use this solution to compute and report the elasticity of labor supply as the percentage change in the $LS$ activity:

```
ELS = 100 * (LS.L - LS0) / LS0;
DISPLAY ELS;
```

As the model is currently constructed, the wage elasticity of labor supply equals zero. This is because the utility function is Cobb-Douglas over goods and leisure, and the consumer's only source of income is labor. As the real wage rises, this increases both the demand for goods (labor supply) and the demand for leisure. These effect exactly balance out and the supply of labor is unchanged.

### 4.53.3.4.6 Exercises 3

(a) One way in which the labor supply elasticity might differ from zero in a model with Cobb-Douglas final demand is if there were income from some other source. Let the consumer be endowed with good $x$ in addition to labor. What $x$ endowment is consistent with a labor supply elasticity equal to 0.15?

(b) A second way to "calibrate" the labor supply elasticity is to change the utility function. We can do this by changing the $s:1$ to $s:SIGMA$, where $SIGMA$ is a scalar value representing the benchmark elasticity of substitution between $x$, $y$ and $L$ in final demand. Modify the program to include $SIGMA$ as a scalar, and find the value for $SIGMA$ consistent with a labor supply elasticity equal to 0.15.
4.53.3.5 The Pure Exchange Model

Partial equilibrium analysis forms the basis of most economics courses at the undergraduate level. In these models we focus on price, supply and demand for a single commodity. The partial equilibrium approach neglects indirect effects, through which changes in the market for one good may influence the market for another good.

In the previous section, we focused on the choices of a single consumer. In the present section, we will explore the implications of interactions between many consumers with heterogeneous preferences. Furthermore, the analysis will explore the potentially important interaction between market prices and income which are determined jointly in a general equilibrium.

The most widely-used graphical framework for multi-agent exchange equilibrium analysis is the Edgeworth-Bowley box as illustrated in Figure 7. In this diagram we model the following economy:

**Figure 7: The Edgeworth-Bowley Box**

- Two types of consumers, denoted $A$ and $B$. We consider $A$ and $B$ to each represent multiple consumers, each with the same endowments and preferences. This assumption is helpful for justifying our assumption of perfectly competitive, price-taking behavior.
- Two commodities, denoted $x$ and $y$.
- Fixed endowments of both goods. The horizontal axis measures the total world endowment of good $X$. The vertical axis measure the total world endowment of good $Y$. Any point in the box then represents an allocation of goods between the two agents. The agent $H$ allocation is measured with respect to the lower left origin. The agent $F$ allocation is measured with respect to the upper right origin.

Each agent has a given initial endowment, here denoted point $E$. Furthermore, we assume that there is no possibility for trade. The indifference curves through point $E$ therefore represent autarchy welfare levels.

The key idea in this model is that trade can improve both agents' welfare. One agent gives some amount good $x$ to the other in return for an amount of good $y$. The "terms of trade", the rate of exchange between $x$ and $y$, is determined by the model. The model illustrates a number of important properties of market economies:
• Trade is mutually beneficial. So long as the transactions are voluntary, neither H nor F will be hurt by engaging in trade.

• Market prices can be used to guide the economy to a Pareto-efficient allocation, a state of affairs in which further mutually-beneficial trades are not possible.

• There is no guarantee that the gains from trade will be "fairly distributed" across consumers. A competitive equilibrium may produce a significant welfare increase for one consumer while have negligible impact on the other.

• There are multiple Pareto-efficient allocations, typically only one of which is a competitive equilibrium. We can use this model to demonstrate that the issues of efficiency and equity can be separated when there is the possibility of lump-sum income transfers between agents.

4.53.3.6 Modeling Pure Exchange with MPSGE

4.53.3.6.1 Example 4: A 2x2 Exchange Model In this program, we examine the simple two good, two agent model of exchange equilibrium. The world endowments for goods $x$ and $y$ are both equal to 1. Six parameters are used to parameterize the model. These are declared as scalars at the top of the program:

```
SCALAR XA AGENT A ENDOWMENT OF X ( 0 < XA < 1) /0.2/ 
YA AGENT A ENDOWMENT OF Y ( 0 < YA < 1) /0.8/ 
THETA_A AGENT A DEMAND SHARE PARAMETER FOR X /0.5/ 
THETA_B AGENT B DEMAND SHARE PARAMETER FOR X /0.8/ 
SIGMA_A AGENT A ELASTICITY PARAMETER /2.0/ 
SIGMA_B AGENT B ELASTICITY PARAMETER /0.5/ 
```

This model is actually simpler than the models presented above because we have no need for production. There are simply two commodities and two consumers. The consumers differ in terms of commodity endowments and preferences. The competitive equilibrium prices are such that supply equals demand for both goods and both agents spend an amount equal to their endowment income.

This model illustrates how to use computed function coefficients. See, for example, $Q:(1-\text{THETA\_A})$ in the $\text{DEMAND:\_A}$ block. Any numeric input field in an MPSGE model may be "computed", provided that the algebraic expression is enclosed within parentheses and legitimate GAMS code.

This model specification uses the default values for reference prices in the demand function blocks. When P:value is not specified in a D:\_I: or O:\_record, P:1 is assumed.

This model uses the more general constant-elasticity-of-substitution utility function.

```gams
$ONTEXT
$MODEL:EXCHANGE
$COMMODITIES:
PX ! EXCHANGE PRICE OF GOOD X
PY ! EXCHANGE PRICE OF GOOD Y
$CONSUMERS:
A ! CONSUMER A
B ! CONSUMER B
$DEMAND:A s:SIGMA\_A
E:PX Q:XA
```
The foregoing sets up the model and computes the competitive equilibrium. After GAMS returns from the solver, we declare and compute some report values.

Absolute levels of income and price returned from a general equilibrium model are not meaningful because a model determines only relative prices. For this reason, we report equilibrium income and price levels in relative terms.

In the final step, we compute an alternative efficient equilibrium, one in which the income levels for A and B are equal. The purpose of this exercise is to demonstrate the second welfare theorem. When incomes are both fixed, the equilibrium remains efficient, but the connection between market prices and endowment income is eliminated.

In GAMS/MPSGE, a variable may be fixed using the GAMS syntax

```
variable.fx = value;
```

as illustrated in this model:

```
A.FX = 1;
B.FX = 1;
$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR TRANSFER IMPLIED TRANSFER FROM A TO B AS A PERCENTAGE OF INCOME;

TRANSFER = 100 * ( A.L - PX.L * XA - PY.L * YA ); PRATIO = PX.L / PY.L;
IRATIO = A.L / B.L;

DISPLAY TRANSFER, PRATIO, IRATIO;
```
4.53.3.6.2 Exercises 4  (a) Set up a separate models which computes the autarchy price ratios for consumers A and B. (You can use one of the earlier models as a starting point.)

(b) Determine parameter values for which the endowment point is the equilibrium point.

(c) Set up a series of computations from which you can sketch the efficiency locus. Draw the Edgeworth box diagram which is consistent with these values.

4.53.3.6.3 Example 5: Import Tariffs and Market Power  The exchange model provides a remarkably useful tool for analyzing issues related to international trade. Formal trade theory is more complicated with the inclusion of separate production technologies. We will present some of those models below. Before going forward, however, we will consider a slight generalization of the 2x2 model exchange model. In this extension, we introduce independent markets for consumers A and B and trade activities which deliver goods from one market to the other.

The set of input parameters largely the same as in the previous example. Two new parameters are ad-valorem tariffs which apply on imports to each of the regions.

```
SCALAR XA  AGENT A ENDOWMENT OF X ( 0 le XA le 1) /0.2/ 
YA  AGENT A ENDOWMENT OF Y ( 0 le YA le 1) /0.8/ 
THETA_A  AGENT A DEMAND SHARE PARAMETER FOR X /0.4/ 
THETA_B  AGENT B DEMAND SHARE PARAMETER FOR X /0.6/ 
SIGMA_A  AGENT A ELASTICITY PARAMETER /1.0/ 
SIGMA_B  AGENT B ELASTICITY PARAMETER /1.0/ 
T_A  AD-VALOREM TARIFF ON IMPORTS TO AGENT A /0.10/ 
T_B  AD-VALOREM TARIFF ON IMPORTS TO AGENT B /0.10/ 
```

The program differs from the previous example in several respects. First, we introduce a separate commodity price for each agent. In the absence of tariffs, these prices are identical.

A second difference is that in this model trade activities deliver goods from one agent to the other. These are denoted \( M\{\text{good}\}\{\text{agent}\} \) for imports of \{good\} to \{agent\}. There are four flows which may be operated in only one direction (the activity levels are non-negative). In terms of initial endowments and preferences, this model has exactly the same economic structure as the previous model.

$ONTEXT
$MODEL:TARIFFS
$SECTORS:
MXA  ! TRADE IN X FROM B TO A 
MXB  ! TRADE IN X FROM A TO B 
MYA  ! TRADE IN Y FROM B TO A 
MYB  ! TRADE IN Y FROM A TO B 

$COMMODITIES:
PXA  ! PRICE OF GOOD X FOR AGENT A 
PYA  ! PRICE OF GOOD Y FOR AGENT A 
PXB  ! PRICE OF GOOD X FOR AGENT B 
PYB  ! PRICE OF GOOD Y FOR AGENT B 

$CONSUMERS:
A  ! CONSUMER A 
B  ! CONSUMER B 

$END
The trade activities each have one input and one output. They simply deliver a good (X or Y) from one agent to the other. The new syntax presented here is specification of an ad-valorem tax. Adding a tax requires two new fields. The first is "A:" which specifies the tax agent, a consumer who collects the tax revenue as part of income. The second is "T:" which specifies the ad-valorem tax rate.

MPSGE permits taxes to be applied on production inputs and outputs but it does not permit taxes on final demand.

The tax applies on a net basis on inputs. For example, if we consider the MXA sector, the price of one unit of input is given by:

\[ P_{x,B} \times (1 + T_a) \]

where \( P_{x,B} \) is the net of tax price of a unit of \( x \) in the agent \( B \) market and \( T_a \) is the ad-valorem tariff rate.

The final portions of the file introduce one use of "MPSGE report variables". In this case, report variables are used to recover a Hicksian money-metric welfare index for each of the agents. We compute the initial, tariff-ridden equilibrium in order to compute the benchmark welfare levels. We then set all tariffs to zero and compute the free-trade equilibrium. Using the final welfare indices and the saved values of the benchmark welfare levels, we are able to report the change in welfare from removing tariff distortions.
$REPORT:
V:WA  W:A
V:WB  W:B

$OFFTEXT
$SYSINCLUDE mpsgeset TARIFFS

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
   WA0  BENCHMARK WELFARE INDEX FOR AGENT A
   WB0  BENCHMARK WELFARE INDEX FOR AGENT B;

WA0 = WA.L;
WB0 = WB.L;
T_A = 0;
T_B = 0;

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
   EVA  HICKSIAN EQUIVALENT VARIATION FOR AGENT A
   EVB  HICKSIAN EQUIVALENT VARIATION FOR AGENT B;

EVA = 100 * (WA.L-WA0)/WA0;
EVB = 100 * (WB.L-WB0)/WB0;
DISPLAY EVA, EVB;

4.53.3.6.4 Exercises 5  (a) Find the "optimal tariff" in this model for agent A, assuming that agent B does not retaliate and leaves her tariff rate at the benchmark level.

(b) Insert the endowment and preference parameters from the previous problem, retaining the same "benchmark" tariff rates. Does free trade benefit both countries? If not, why not?

4.53.4 Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae

Notes prepared for GAMS General Equilibrium Workshop held December, 1995 in Boulder Colorado

Author

Thomas F. Rutherford, Department of Economics, University of Colorado

Date
December, 1995
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1355

4.53.4.1 The Basics

In many economic textbooks the constant elasticity of substitution (CES) utility function is defined as:

\[ U(x, y) = (\alpha x^\rho + (1 - \alpha) y^\rho)^{\frac{1}{\rho}}, \]

It is a fairly routine but tedious calculus exercise to demonstrate that the associated demand functions are:

\[ x(p_x, p_y, M) = \frac{\alpha}{p_x} \left( \frac{M}{\alpha \sigma p_x^{1-\sigma} + (1 - \alpha) \sigma p_y^{1-\sigma}} \right)^{\frac{1}{\sigma}}, \]

and

\[ y(p_x, p_y, M) = \frac{1 - \alpha}{p_y} \left( \frac{M}{\alpha \sigma p_x^{1-\sigma} + (1 - \alpha) \sigma p_y^{1-\sigma}} \right)^{\frac{1}{\sigma}}, \]

The corresponding indirect utility function has is:

\[ V(p_x, p_y, M) = \left( \frac{\alpha \sigma p_x^{1-\sigma} + (1 - \alpha) \sigma p_y^{1-\sigma}}{1} \right)^{\frac{1}{1 - \sigma}}, \]

Note that \( U(x, y) \) is linearly homogeneous:

\[ U(\lambda x, \lambda y) = \lambda U(x, y) \]

This is a convenient cardinalization of utility, because percentage changes in \( U \) are equivalent to percentage Hicksian equivalent variations in income.

Because \( U \) is linearly homogeneous, \( V \) is homogeneous of degree one in \( M \) and degree \(-1\) in \( p \).

In the representation of technology, we have an analogous set of relationships, based on the cost and compensated demand functions. If we have a CES production function of the form:

\[ y(K, L) = \gamma (\alpha K^\rho + (1 - \alpha) L^\rho)^{\frac{1}{\rho}}, \]

the unit cost function then has the form:

\[ c(p_K, p_L) = \left( \frac{1}{\gamma} \right) \left( \alpha \sigma p_K^{1-\sigma} + (1 - \alpha) \sigma p_L^{1-\sigma} \right)^{\frac{1}{1 - \sigma}}, \]

and associated demand functions are:

\[ K(p_K, p_L) = \left( \frac{y}{\gamma} \right) \left( \frac{\alpha \gamma c(p_K, p_L)}{p_K} \right)^{\sigma}, \]

and

\[ L(p_K, p_L) = \left( \frac{y}{\gamma} \right) \left( \frac{(1 - \alpha) \gamma c(p_K, p_L)}{p_L} \right)^{\sigma}. \]

In most large-scale applied general equilibrium models, we have many function parameters to specify with relatively few observations. The conventional approach is to *calibrate* functional parameters to a single benchmark equilibrium. For example, if we have benchmark estimates for output, labor, capital inputs and factor prices, we calibrate function coefficients by inverting the factor demand functions:

\[ \theta = \frac{p_K}{p_K + p_L}, \quad \rho = \frac{\sigma - 1}{\sigma}, \quad \alpha = \theta \bar{K}^\frac{\sigma - 1}{\rho}, \]

and

\[ \gamma = \bar{y} \left[ \alpha \bar{K}^\rho + (1 - \alpha) \bar{L}^\rho \right]^\frac{1}{\rho}. \]
### 4.53.4.2 The Calibrated Share Form

Calibration formulae for CES functions are messy and difficult to remember. Consequently, the specification of function coefficients is complicated and error-prone. For applied work using calibrated functions, it is much easier to use the “calibrated share form” of the CES function. In the calibrated form, the cost and demand functions explicitly incorporate:

- benchmark factor demands
- benchmark factor prices
- the elasticity of substitution
- benchmark cost
- benchmark output
- benchmark value shares

In this form, the production function is written:

\[ y = \bar{y} \left[ \theta \left( \frac{K}{\bar{K}} \right)^{\rho} + (1 - \theta) \left( \frac{L}{\bar{L}} \right)^{\rho} \right]^{\frac{1}{\rho}} \]

The only calibrated parameter, \( \theta \), represents the value share of capital at the benchmark point. The corresponding cost functions in the calibrated form is written:

\[ c(p_K, p_L) = \tilde{c} \left[ \theta \left( \frac{p_K}{\tilde{p}_K} \right)^{1-\sigma} + (1 - \theta) \left( \frac{p_L}{\tilde{p}_L} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}} \]

where \( \tilde{c} = \tilde{p}_L \tilde{L} + \tilde{p}_K \tilde{K} \) and the compensated demand functions are:

\[ K(p_K, p_L, y) = \bar{K} \frac{y}{\tilde{c}} \left( \frac{p_K}{\tilde{p}_K} \right)^{\frac{\sigma}{\rho}} \]

and

\[ L(p_K, p_L, y) = \bar{L} \frac{y}{\tilde{c}} \left( \frac{p_L}{\tilde{p}_L} \right)^{\frac{\sigma}{\rho}}. \]

Normalizing the benchmark utility index to unity, the utility function in calibrated share form is written:

\[ U(x, y) = \left[ \theta \left( \frac{x}{\bar{x}} \right)^{\rho} + (1 - \theta) \left( \frac{y}{\bar{y}} \right)^{\rho} \right]^{\frac{1}{\rho}} \]

Defining the unit expenditure function as:

\[ e(p_x, p_y) = \left[ \theta \left( \frac{p_x}{\bar{p}_x} \right)^{1-\sigma} + (1 - \theta) \left( \frac{p_y}{\bar{p}_y} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}} \]

the indirect utility function is:

\[ V(p_x, p_y, M) = \frac{M}{Me(p_x, p_y)} \]
and the demand functions are:

\[ x(p_x, p_y, M) = \bar{x} V(p_x, p_y, M) \left( \frac{e(p_x, p_y)p_x}{p_x} \right)^{\sigma} \]

and

\[ y(p_x, p_y, M) = \bar{y} V(p_x, p_y, M) \left( \frac{e(p_x, p_y)p_y}{p_y} \right)^{\sigma} \]

The calibrated form extends directly to the n-factor case. An n-factor production function is written:

\[ y = f(x) = \bar{y} \left[ \sum_i \theta_i \left( \frac{x_i}{\bar{x}_i} \right)^{\rho} \right]^{\frac{1}{\rho}} \]

and has unit cost function:

\[ C(p) = \bar{C} \left[ \sum_i \theta_i \left( \frac{p_i}{\bar{p}_i} \right)^{1-\sigma} \right] \frac{1}{1-\sigma} \]

and compensated factor demands:

\[ x_i = \bar{x}_i \frac{y}{\bar{y}} \left( \frac{C \bar{p}_i}{C p_i} \right)^{\sigma} \]

4.53.4.3 Excercises

(i) Show that given a generic CES utility function:

\[ U(x, y) = (\alpha \rho + (1 - \alpha) y^\rho)^{\frac{1}{\rho}} \]

can be represented in share form using:

\[ \bar{x} = 1, \quad \bar{y} = 1, \quad \bar{p}_x = t\alpha, \quad \bar{p}_y = t(1 - \alpha), \quad \bar{M} = t. \]

for any value of \( t > 0 \).

(ii) Consider the utility function defined:

\[ U(x, y) = (x - a)^\alpha (y - b)^{1-\alpha} \]

A benchmark demand point with both prices equal and demand for \( y \) equal to twice the demand for \( x \). Find values for which are consistent with optimal choice at the benchmark. Select these parameters so that the income elasticity of demand for \( x \) at the benchmark point equals 1.1.

(iii) Consider the utility function:

\[ U(x, L) = (\alpha L^\rho + (1 - \alpha) x^\rho)^{\frac{1}{\rho}} \]
which is maximized subject to the budget constraint:

\[ p_k x = M + \omega (L - L) \]

in which \( M \) is interpreted as non-wage income, \( \omega \) is the market wage rate. Assume a benchmark equilibrium in which prices for \( x \) and \( L \) are equal, demands for \( x \) and \( L \) are equal, and non-wage income equals one-half of expenditure on \( x \). Find values of \( \alpha \) and \( \rho \) consistent with these choices and for which the price elasticity of labor supply equals 0.2.

(iv) Consider a consumer with CES preferences over two goods. A price change makes the benchmark consumption bundle unaffordable, yet the consumer is indifferent. Graph the choice. Find an equation which determines the elasticity of substitution as a function of the benchmark value shares. (You can write down the equation, but it cannot be solved in closed form.)

(v) Consider a model with three commodities, \( x \), \( y \), and \( z \). Preferences are CES. Benchmark demands and prices are equal for all goods. Find demands for \( x \), \( y \) and \( z \) for a doubling in the price of \( x \) as a function of the elasticity of substitution.

(iv) Consider the same model in the immediately preceding question, except assume that preferences are instead given by:

\[ U(x, y, z) = \left( \beta \min(x, y)^\rho + (1 - \beta)z^\rho \right)^{\frac{1}{\rho}} \]

Determine \( \beta \) from the benchmark, and find demands for \( x \), \( y \) and \( z \) if the price of \( x \) doubles.

4.53.4.4 Flexibility and Non-Separable CES functions

We let \( \pi_i \) denote the user price of the \( i \)th input, and let \( x_i(\pi) \) be the cost-minimizing demand for the \( i \)th input. The reference price and quantities are \( \bar{\pi} \) and \( \bar{x}_i \). One can think of set \( i \) as \( \{K, L, E, M\} \) but the methods we employ may be applied to any number of inputs. Define the reference cost, and reference value share for \( i \)th input by \( \bar{C} \) and \( \theta_i \), where

\[ \bar{C} \equiv \sum_i \bar{\pi}_i \bar{x}_i \]

and

\[ \theta_i \equiv \frac{\pi_i \bar{x}_i}{\bar{C}}. \]

The single-level constant elasticity of substitution cost function in "calibrated share form" is written:

\[ C(\pi) = \bar{C} \left( \sum_i \theta_i \left( \frac{\pi_i}{\bar{\pi}_i} \right)^{1-\sigma} \right)^{\frac{1}{1-\sigma}} \]

Compensated demands may be obtained from Shephard’s lemma:

\[ x_i(\pi) = \frac{\delta C}{\delta \pi_i} \equiv C_i = \bar{x}_i \left( \frac{C(\pi) \bar{\pi}_i}{\bar{C} \pi_i} \right)^\sigma \]
Cross-price Allen-Uzawa elasticities of substitution (AUES) are defined as:

$$\sigma_{ij} \equiv \frac{C_{ij}C}{C_i C_j}$$

where

$$C_{ij} = \frac{\delta^2 C(\pi)}{\delta \pi_i \delta \pi_j} = \frac{\delta x_i}{\delta \pi_j} = \frac{\delta x_j}{\delta \pi_i}$$

For single-level CES functions:

$$\sigma_{ij} = \sigma \quad \forall i \neq j$$

The CES cost function exhibits homogeneity of degree one, hence Euler's condition applies to the second derivatives of the cost function (the Slutsky matrix):

$$\sum_j C_{ij}(\pi)(\pi_j) = 0$$

or, equivalently:

$$\sum_j \sigma_{ij}(\theta_j) = 0$$

The Euler condition provides a simple formula for the diagonal AUES values:

$$\sigma_{ii} = -\sum_{j \neq i} \sigma_{ij} \theta_j \theta_i$$

As an aside, note that convexity of the cost function implies that all minors of order 1 are negative, i.e. $$\sigma_{ii} < 0 \quad \forall i$$. Hence, there must be at least one positive off-diagonal element in each row of the AUES or Slutsky matrices. When there are only two factors, then the off-diagonals must be negative. When there are three factors, then only one pair of negative goods may be complements.

Let:

- $k$ be the reference the index of second-level nest
- $s_{ik}$ denote the fraction of good $i$ inputs assigned to the $k$th nest
- $\omega_k$ denote the benchmark value share of total cost which enters through the $k$th nest
- $\gamma$ denote the top-level elasticity of substitution
- $\sigma^k$ denote the elasticity of substitution in the $k$th aggregate
- $p_k(\pi)$ denote the price index associated with aggregate $k$, normalized to equal unity in the benchmark, i.e.:

$$p_k(\pi) = \left[ \frac{\sum_i s_{ik} \theta_i}{\omega_k \left( \frac{\pi_i}{\pi_i^*} \right)^{1-\sigma^k}} \right]^{1-\sigma^k}$$
The two-level nested, nonseparable constant-elasticity-of-substitution (NNCES) cost function is then defined as:

\[ C(\pi) = \bar{C} \left( \sum_k \omega_k p_k(\pi)^{1-\gamma} \right)^{\gamma} \]

Demand indices for second-level aggregates are needed to express demand functions in a compact form. Let \( z_k(\pi) \) denote the demand index for aggregate \( k \), normalized to unity in the benchmark; i.e.

\[ z_k(\pi) = \left( \frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^{\gamma} \]

Compensated demand functions are obtained by differentiating \( C(\pi) \). In this derivative, one term arise for each nest in which the commodity enters, so:

\[ x_i(\pi) = \bar{x}_i \sum_k z_k(\pi) \left( \frac{p_k(\pi)\bar{\pi}_i}{\pi_i} \right)^{\sigma^k} = \bar{x}_i \sum_k \left( \frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^{\gamma} \left( \frac{p_k(\pi)\bar{\pi}_i}{\pi_i} \right)^{\sigma^k} \]

Simple differentiation shows that benchmark cross-elasticities of substitution have the form:

\[ \sigma_{ij} = \gamma + \sum_k \left( \sigma^k - \gamma \right) s_{ik} s_{jk} \omega_k \]

Given the benchmark value shares \( \theta_i \) and the benchmark cross-price elasticities of substitution, \( \sigma_{ij} \), we can solve for values of \( s_{ik} \), \( \omega_k \), \( \sigma^k \) and \( \gamma \). We compute these parameters using a constrained nonlinear programming algorithm, CONOPT, which is available through GAMS, the same programming environment in which the equilibrium model is specified. Perroni and Rutherford (EER, 1994) prove that calibration of the NNCES form is possible for arbitrary dimensions whenever the given Slutsky matrix is negative semi-definite. The two-level (NxN) function is flexible for three inputs; and although we have not proven that it is flexible for 4 inputs, the only difficulties we have encountered have resulted from indefinite calibration data points.

Two GAMS programs are listed below. The first illustrates two analytic calibrations of the three-factor cost function. The second illustrates the use of numerical methods to calibrate a four-factor cost function.

**4.53.4.5 Two NNCES calibrations for a 3-input cost functions**

* * * Model-specific data defined here: * * *

SET I Production input aggregates / A,B,C /; ALIAS (I,J);

PARAMETER

THETA(I) Benchmark value shares / A 0.2, B 0.5, C 0.3 /

AUES(I,J) Benchmark cross-elasticities (off-diagonals) /

A,B 2
A,C -0.05
B,C 0.5 /

* * *
* Use an analytic calibration of the three-factor CES cost function:

```
ABORT$(CARD(I) NE 3) "Error: not a three-factor model!"
```

* Fill in off-diagonals:

```
AUES(I,J)$AUES(J,I) = AUES(J,I);
```

* Verify that the cross elasticities are symmetric:

```
ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?"
```

* Check that all value shares are positive:

```
ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;
```

* Fill in the elasticity matrices:

```
AEsu(I,I) = 0; AEsu(I,I) = -SUM(J, AEsu(I,J)*THETA(J))/THETA(I); DISPLAY AEsu;
```

```
SET N Potential nesting /N1*N3/
K(N) Nesting aggregates used in the model
I1(I) Good fully assigned to first nest
I2(I) Good fully assigned to second nest
I3(I) Good split between nests;
```

```
SCALAR ASSIGNED /0/;
```

```
PARAMETER
ESUB(*,*) Alternative calibrated elasticities
SHR(*,I,N) Alternative calibrated shares
SIGMA(N) Second level elasticities
S(I,N) Nesting assignments (in model)
GAMMA Top level elasticity (in model);
```

* First the Leontief structure:

```
ESUB("LTF","GAMMA") = SMAX((I,J), AEsu(I,J));
ESUB("LTF",N) = 0;
LOOP((I,J)$((AEsu(I,J) EQ ESUB("LTF","GAMMA"))*(NOT ASSIGNED)),
    I1(I) = YES;
    I2(J) = YES;
    ASSIGNED = 1;
);
I3(I) = YES$((NOT I1(I))*(NOT I2(I)));
DISPLAY I1,I2,I3;
LOOP((I1,I2,I3),
    SHR("LTF",I1,"N1") = 1;
    SHR("LTF",I2,"N2") = 1;
    SHR("LTF",I3,"N1") = THETA(I1)*(1-AUES(I1,I3)/AUES(I1,I2)) /
        ( 1 - THETA(I3) * (1-AUES(I1,I3)/AUES(I1,I2)) )
    SHR("LTF",I3,"N2") = THETA(I2)*(1-AUES(I2,I3)/AUES(I1,I2)) /
        ( 1 - THETA(I3) * (1-AUES(I2,I3)/AUES(I1,I2)) )
);
```
ABORT$(SMIN((I,N), SHR("LTF",I,N)) LT 0) "Benchmark AUES is indefinite."

* Now, the CES function:

ESUB("CES","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("CES","N1") = 0;
LOOP((I1,I2,I3),
    SHR("CES",I1,"N1") = 1;
    SHR("CES",I2,"N2") = 1;
    ESUB("CES","N2") = (AUES(I1,I2)*AUES(I1,I3)-AUES(I2,I3)*AUES(I1,I1)) / (AUES(I1,I3)-AUES(I1,I1));
    SHR("CES",I3,"N1") = (AUES(I1,I2)-AUES(I1,I3)) / (AUES(I1,I2)-AUES(I1,I1));
    SHR("CES",I3,"N2") = 1 - SHR("CES",I3,"N1");
);
ABORT$(SMIN(N, ESUB("CES",N)) LT 0) "Benchmark AUES is indefinite?";
ABORT$(SMIN((I,N), SHR("CES",I,N)) LT 0) "Benchmark AUES is indefinite?"

PARAMETER PRICE(I) PRICE INDICES USING TO VERIFY CALIBRATION
AUESCHK(*,I,J) CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext
$MODEL:CHKCALIB

$SECTORS:
Y ! PRODUCTION FUNCTION
D(I)

$COMMODITIES:
PY ! PRODUCTION FUNCTION OUTPUT
P(I) ! FACTORS OF PRODUCTION
PFX ! AGGREGATE PRICE LEVEL

$CONSUMERS:
RA

$PROD:Y s:GAMMA K.TL:SIGMA(K)
O:PY Q:1
I:P(I)#(K) Q:(THETA(I)*S(I,K)) K.TL:

$PROD:D(I)
O:P(I) Q:THETA(I)
I:PFX Q:(THETA(I)*PRICE(I))

$DEMAND:RA
D:PFX
E:PFX Q:2
E:PY Q:-1

$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

SCALAR DELTA /1.E-5/;

SET FUNCTION /LTF, CES/
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1363

ALIAS (I,II);

LOOP(FUNCTION,
    K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
    GAMMA = ESUB(FUNCTION,“GAMMA”);
    SIGMA(K) = ESUB(FUNCTION,K);
    S(I,K) = SHR(FUNCTION,I,K);

    LOOP(II,
        PRICE(J) = 1; PRICE(II) = 1 + DELTA;
        $INCLUDE CHKCALIB.GEN
        SOLVE CHKCALIB USING MCP;
        AUUESCHK(FUNCTION,J,II) = (D.L(J)-1) / (DELTA*THETA(II));
    ));

    AUUESCHK(FUNCTION,I,J) = AUUESCHK(FUNCTION,I,J) - AUES(I,J);
    DISPLAY AUUESCHK;
    *
    * Evaluate the demand functions:
    *
    $LIBINCLUDE qadplot
    SET PR Alternative price levels /PR0*PR10/;

    PARAMETER
        DEMAND(FUNCTION,I,PR) Demand functions
        DPlot(PR,FUNCTION) Plotting output array;

    LOOP(II,
        LOOP(FUNCTION,
            K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
            GAMMA = ESUB(FUNCTION,“GAMMA”);
            SIGMA(K) = ESUB(FUNCTION,K);
            S(I,K) = SHR(FUNCTION,I,K);
            LOOP(PR,
                PRICE(J) = 1;
                PRICE(II) = 0.2 * ORD(PR);
                $INCLUDE CHKCALIB.GEN
                SOLVE CHKCALIB USING MCP;
                DEMAND(FUNCTION,II,PR) = D.L(II);
                DPlot(PR,FUNCTION) = D.L(II);
            );
    );

    $LIBINCLUDE qadplot DPlot PR FUNCTION
    );
    DISPLAY DEMAND;

4.53.4.6 A Comparison of Locally-Identical Functions

A Comparison of Locally-Identical Functions
4.53.4.7 Numerical calibration of NNCES given KLEM elasticities

SET I Production input aggregates / K, L, E, M/; ALIAS (I,J);

* Model-specific data defined here:

PARAMETER
 THETA(I) Benchmark value shares /K 0.2, L 0.4, E 0.05, M 0.35/

 AUES(I,J) Benchmark cross-elasticities (off-diagonals) /
        K.L 1
        K.E -0.1
        K.M 0
        L.E 0.3
        L.M 0
        E.M 0.1 /;

* Model-specific data defined here:
4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1365
SCALAR EPSILON  Minimum value share tolerance /0.001/;

*  Fill in off-diagonals:

AUES(I,J)$AUES(J,I) = AUES(J,I);

*  Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*  Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:" ,THETA;

*  Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

*  Define variables and equations for NNCES calibration:

SET N Nests within the two-level NNCES function /N1*N4/,
      K(N) Nests which are in use;

VARIABLES
   S(I,N) Fraction of good I which enters through nest N,
   SHARE(N) Value share of nest N,
   SIGMA(N) Elasticity of substitution within nest N,
   GAMMA Elasticity of substitution at the top level,
   OBJ Objective function;

POSITIVE VARIABLES S, SHARE, SIGMA, GAMMA;

EQUATIONS
   SDEF(I) Nest shares must sum to one,
   TDEF(N) Nest share in total cost,
   ELAST(I,J) Consistency with given AUES values,
   OBJDEF Maximize concentration;

ELAST(I,J)$(ORD(I) GT ORD(J))..

   AUES(I,J) =E=  GAMMA +
   SUM(K, (SIGMA(K)-GAMMA) * S(I,K) * S(J,K)/SHARE(K));

TDEF(K)..
   SHARE(K) =E= SUM(I, THETA(I) * S(I,K));

SDEF(I)..
   SUM(N, S(I,N)) =E= 1;

*  Maximize concentration at the same time keeping the elasticities
*  to be reasonable:

OBJDEF..
   OBJ =E= SUM((I,K),S(I,K)*S(I,K))
   - SQR(GAMMA) - SUM(K, SQR(SIGMA(K)));

MODEL CESCALIB /ELAST, TDEF, SDEF, OBJDEF/;
* Apply some bounds to avoid divide by zero:

\[
SHARE.LO(N) = EPSILON;
\]

** SCALAR SOLVED Flag for having solved the calibration problem /0/
** MINSHR Minimum share in candidate calibration;

** SET TRIES Counter on the number of attempted calibrations /T1*T10/;
** We use the random number generator to select starting points,
** so it is helpful to initialize the seed so that the results
** will be reproducible:

** OPTION SEED=0;

** LOOP(TRIES$(NOT SOLVED),
** Initialize the set of active nests and the bounds:

\[
K(N) = YES;
S.LO(I,N) = 0; \quad S.UP(I,N) = 1;
SHARE.LO(N) = EPSILON; \quad SHARE.UP(N) = 1;
SIGMA.LO(N) = 0; \quad SIGMA.UP(N) = +INF;
\]

** Install a starting point:

\[
SHARE.L(K) = \max(\text{UNIFORM}(0,1), EPSILON);
S.L(I,K) = \text{UNIFORM}(0,1);
\]
\[
\text{GAMMA.L} = \text{UNIFORM}(0,1);
\]
\[
\text{SIGMA.L}(K) = \text{UNIFORM}(0,1);
\]

** Drop any basis information so that we start from scratch:

\[
SDEF.M(I) = 0; \quad TDEF.M(K) = 0; \quad ELAST.M(I,J) = 0;
\]

** SOLVE CESCALIB USING NLP MAXIMIZING OBJ;
** SOLVED = 1$(CESCALIB.MODELSTAT LE 2);
** We have a solution -- now see if it is not on a bound:

** IF (SOLVED,
** \[
\text{MINSHR} = \text{SMIN}(K, SHARE.L(K)) - EPSILON;
\]
** IF (MINSHR EQ 0,
** Drop nests which have shares equal to EPSILON in the current
** solution:

\[
K(N)$(SHARE.L(N) EQ EPSILON) = NO;
S.FX(I,N)$(NOT K(N)) = 0;
SHARE.FX(N)$(NOT K(N)) = 0;
SIGMA.FX(N)$(NOT K(N)) = 0;
\]
DISPLAY "Recalibrating with the following nests:", K;

SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

IF (CESCALIB.MODELSTAT GT 2, SOLVED = 0);
    MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;
    IF (MINSHR EQ 0, SOLVED = 0);
);

IF (SOLVED,
    DISPLAY "Function calibrated:", GAMMA.L, SIGMA.L, SHARE.L, S.L;
ELSE
    DISPLAY "Function calibration fails!";
);

*==========================================================================

Solution from MINOS5 obtained on the second try, following an
ITERATION INTERRUPT on the first:

----  151 Function calibrated:

----  151 VARIABLE GAMMA.L = 0.300 Elasticity of
         substitution at the
         top level

----  151 VARIABLE SIGMA.L Elasticity of substitution within nest N
         N3 7.804

----  151 VARIABLE SHARE.L Value share of nest N
         N1 0.604,  N2 0.266,  N3 0.030,  N4 0.100

----  151 VARIABLE S.L Fraction of good I which enters through
         nest N
         N1 0.797  0.069  0.133
         N2 0.960  0.040
         N3  E 1.000
         N4  0.630  0.304  0.067

*==========================================================================

The following solution is obtained by CONOPT on the second try, following a LOCALLY INFEASIBLE
termination on the first problem. Notice that it is identical to the MINOS5 solution except that the
nesting assignments have been permuted:
---- 149 Function calibrated:

---- 149 VARIABLE GAMMA.L = 0.300 Elasticity of substitution at the top level

---- 149 VARIABLE SIGMA.L Elasticity of substitution within nest N

N4 7.804

---- 149 VARIABLE SHARE.L Value share of nest N

N1 0.100, N2 0.604, N3 0.266, N4 0.030

---- 149 VARIABLE S.L Fraction of good I which enters through nest N

<table>
<thead>
<tr>
<th></th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>0.133</td>
<td>0.797</td>
<td>0.069</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>0.960</td>
<td>0.040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>0.067</td>
<td>0.630</td>
<td>0.304</td>
<td></td>
</tr>
</tbody>
</table>

==========================================================================

$OFFTEXT

PARAMETER PRICE(I) PRICE INDICES USING TO VERIFY CALIBRATION
AUESCHK(I,J) CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext

$MODEL:CHKCALIB

$SECTORS:

Y ! PRODUCTION FUNCTION
D(I)

$COMMODITIES:

PY ! PRODUCTION FUNCTION OUTPUT
P(I) ! FACTORS OF PRODUCTION
PFX ! AGGREGATE PRICE LEVEL

$CONSUMERS:

RA

$PROD:Y s:GAMMA.L K.TL:SIGMA.L(K)
O:PY Q:1
I:P(I)#(K) Q:(THETA(I)*S.L(I,K)) K.TL:

$PROD:D(I)
4.53.4.8 Calibrating Labor Supply and Savings Demand

This material was published in the MUG newsletter, 8/95.

Following Ballard, Fullerton, Shoven and Whalley (BFSW), we consider a representative agent whose utility is based upon current consumption, future consumption and current leisure. Changes in “future consumption” in this static framework are associated with changes in the level of savings. There are three prices which jointly determine the price index for future consumption. These are:

\[ P_I \] the composite price index for investment goods

\[ P_K \] the composite rental price for capital services

\[ P_C \] the composite price of current consumption.

All of these prices equal unity in the benchmark equilibrium.

Capital income in each future year finances future consumption, which is expected to cost the same as in the current period, \( P_C \) (static expectations). The consumer demand for savings therefore depends not only on \( P_I \), but also on \( P_K \) and \( P_C \), namely:

\[ P_S = \frac{P_I P_C}{P_K} \]

The price index for savings is unity in the benchmark period. In a counter-factual equilibrium, however, we would expect generally that \( P_S \neq P_I \). When these price indices are not equal, there is a “virtual tax payment” associated with savings demand.
Following BFSW, we adopt a nested constant-elasticity-of-substitution function to represent preferences. In this function, at the top level demand for savings (future consumption) trades off with a second CES aggregate of leisure and current consumption. These preferences can be summarized with the following expenditure function:

\[
P_U = \left[ \alpha P_H^{1-\sigma_S} + (1 - \alpha)P_H^{1-\sigma_S} \right]^{\frac{1}{1-\sigma_S}}
\]

Preferences are homothetic, so we have defined \( P_U \) as a linearly homogeneous cost index for a unit of utility. We conveniently scale this price index to equal unity in the benchmark. In this definition, \( \alpha \) is the benchmark value share for current consumption (goods and leisure). \( P_H \) is a compositive price for current consumption defined as:

\[
P_H = \left[ \beta P_I^{1-\sigma_L} + (1 - \beta)P_C^{1-\sigma_L} \right]^{\frac{1}{1-\sigma_L}}
\]

in which \( \beta \) is the benchmark value share for leisure within current consumption. Demand functions can be written as follows:

\[
S = S_0 \left( \frac{P_U}{P_F} \right)^{\frac{\sigma_S}{1-\sigma_S}} \frac{I}{I_0 P_U},
\]

\[
C = C_0 \left( \frac{P_H}{P_C} \right)^{\frac{\sigma_L}{1-\sigma_L}} \left( \frac{P_U}{P_H} \right)^{\frac{\sigma_S}{1-\sigma_S}} \frac{I}{I_0 P_U},
\]

and

\[
\ell = \ell_0 \left( \frac{P_H}{P_L} \right)^{\frac{\sigma_L}{1-\sigma_L}} \left( \frac{P_U}{P_H} \right)^{\frac{\sigma_S}{1-\sigma_S}} \frac{I}{I_0 P_U},
\]

Demands are written here in terms of their benchmark values (\( S_0, C_0 \) and \( \ell_0 \)) and current and benchmark income (\( I \) and \( I_0 \)).

There are four components in income. The first is the value of labor endowment (\( E \)), defined inclusive of leisure. The second is the value of capital endowment (\( K \)). The third is all other income (\( M \)). The fourth is the value of "virtual tax revenue" associated with differences between the shadow price of savings and the cost of investment.

\[
I = P_L E + P_K K + M + (P_S - P_I)S
\]

The following parameter values are specified exogenously:

1. \( \zeta = 1.75 \) is the ratio of labor endowment:

\[
\zeta \equiv \frac{E}{L_0}
\]

where \( L_0 \) is the benchmark labor supply. Given \( \zeta \) and \( L_0 \) we have:

\[
\ell_0 = L_0 (\zeta - 1)
\]

2. \( \xi = 0.15 \) is the uncompensated elasticity of labor supply with respect to the net of tax wage, i.e.

\[
\xi = \frac{\delta L}{\delta P_L} \frac{P_L}{L} = \frac{\delta(E - \ell)}{\delta P_L} \frac{P_L}{L} = -\frac{\delta \ell}{\delta P_L} \frac{P_L}{L}
\]
3. $\eta = 0.4$ is the elasticity of savings with respect to the return to capital:

\[
\eta \equiv \frac{\delta S}{\delta P_K} \frac{S}{P_K}
\]

Shephard's lemma applied at benchmark prices provides the following identities which are helpful in deriving expressions for $\eta$ and $\xi$:

\[
\frac{\delta P_U}{\delta P_H} = \alpha, \quad \frac{\delta P_U}{\delta P_S} = 1 - \alpha, \quad \frac{\delta P_H}{\delta P_L} = \beta, \quad \frac{\delta P_H}{\delta P_C} = 1 - \beta,
\]

It is then a relatively routine application of the chain rule to show that:

\[
\xi = (\zeta - 1) \left[ \sigma_L + \beta(\sigma_S - \sigma_L) - \alpha \beta(\sigma_S - 1) - \frac{E}{I_0} \right]
\]

and

\[
\eta = \sigma_S \alpha + \frac{K}{I_0}
\]

The expression for $\eta$ does not involve $\sigma_L$, so we may first solve for $\sigma_S$ and use this value in determining $\sigma_L$:

\[
\sigma_S = \frac{\eta - \frac{K}{I_0}}{\alpha}
\]

and

\[
\alpha_L = \frac{\xi - \sigma_S \beta(1 - \alpha) - \alpha \beta + \frac{E}{I_0}}{1 - \beta}
\]

### 4.53.4.9 A Maquette Illustrating Labor Supply and Savings Demand Calibration

* Exogenous elasticity:

SCALAR XI UNCOMPENSATED ELASTICITY OF LABOR SUPPLY /0.15/,
ETA ELASTICITY OF SAVINGS WRT RATE OF RETURN /0.40/,
ZETA RATIO OF LABOR ENDOWMENT TO LABOR SUPPLY /1.75/;

* Benchmark data:

SCALAR C0 CONSUMPTION /2.998845E+2/,
S0 SAVINGS /70.02698974/,
LS0 LABOR SUPPLY / 2.317271E+2/,
K0 CAPITAL INCOME /93.46960577/,
PLO MARGINAL WAGE /0.60000000/;

* Calibrated parameters:

SCALAR EL0 LABOR ENDOWMENT
L0 LEISURE DEMAND
M0 NON-WAGE INCOME
I EXTENDED GROSS INCOME
ETAMIN SMALLEST PERMISSIBLE VALUE FOR ETA,
XIMIN SMALLEST PERMISSIBLE VALUE FOR XI,
ALPHA CURRENT CONSUMPTION VALUE SHARE
BETA LEISURE VALUE SHARE IN CURRENT CONSUMPTION
SIGMA_L ELASTICITY OF SUBSTITUTION WITHIN CURRENT CONSUMPTION
SIGMA_S ELASTICITY OF SUBSTITUTION - SAVINGS VS CURRENT CONSUMPTION
TS SAVINGS PRICE ADJUSTMENT;

* Convert labor supply into net of tax units:
LS0 = LS0 * PL0;
* Labor endowment (exogenous):
ELO = ZETA * LS0;
* Leisure demand:
L0 = ELO - LS0;
* Non-labor, non-capital income:
M0 = C0 + S0 - LS0 - K0;
* Extended gross income:
I = L0 + C0 + S0;
* Leisure share of current consumption:
BETA = L0 / (C0 + L0);
* Current consumption value share:
ALPHA = (L0 + C0) / I;
* Calibrated elasticity:
SIGMA_S = (ETA - K0 / I) / ALPHA;
ETAMIN = K0 / I;
ABORT$(SIGMA_S LT 0) " Error: cannot calibrate SIGMA_S", ETAMIN;
* Calibrated elasticity of substitution between leisure and consumption:
SIGMA_L = (XI*(LS0/L0)-SIGMA_S*BETA*(1-ALPHA)-ALPHA*BETA+EL0/I)/(1-BETA);
XIMIN = -(L0/LS0) * (- SIGMA_S * BETA * (1-ALPHA) - ALPHA*BETA + EL0/I);
ABORT$(SIGMA_L LT 0) " Error: cannot calibrate SIGMA_L", XIMIN;
DISPLAY "Calibrated elasticities:", SIGMA_S, SIGMA_L;

$ONTEXT

$MODEL:CHKCAL

$COMMODITIES:
  PL
  PK
  PC
  PS

$SECTORS:
  Y
  S

$CONSUMERS:
  RA
$PROD: Y
0: PC Q: (K0 + LS0 - S0)
1: PL Q: (LS0 - S0)
1: PK Q: K0

$PROD: S
0: PS A: RA T: TS
1: PL

$DEMAND: RA s: SIGMA_S a: SIGMA_L
E: PC Q: M0
E: PL Q: ELO
E: PK Q: K0
D: PS Q: S0
D: PC Q: C0 a:
D: PL Q: L0 a:

$OFFTEXT
$SYSINCLUDE mpsgeset CHKCAL
S.L = S0;
TS = 0;

* VERIFY THE BENCHMARK:

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;

* CHECK THE LABOR SUPPLY ELASTICITY:

PL.L = 1.001;

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;

* Compute induced changes in labor supply using the labor market
* "marginal", PL.M. This marginal returns the net excess supply of
* labor at the given prices. We started from a balanced benchmark,
* with no change in labor demand (the iteration limit was zero).
* Hence, PL.M returns the magnitude of the change in labor supply.
* We multiply by the benchmark wage (1) and divide by the benchmark
* labor supply (LS0) to produce a finite difference approximation
* of the elasticity:

DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", XI;
XI = (PL.M / 0.001) * (1 / LS0);
DISPLAY XI;

PL.L = 1.0;

* CHECK THE ELASTICITY OF SAVINGS WRT RENTAL RATE OF CAPITAL:

PK.L = 1.001;
PS.L = 1 / 1.001;
TS = 1 / 1.001 - 1;

CHKCAL.ITERLIM = 0;
Compute elasticity of savings with respect to the rental rate of capital. This requires some recursion in order to account for the effect of changes in savings on effective income. When PK increases, PS declines -- there is an effective "subsidy" for saving, paid from consumer income. In order to obtain a difference approximation for the elasticity of savings response, we need to make sure the virtual tax payments are properly handled. In the MPSGE model, this means that the level value for S must be adjusted so that it exactly equals the savings. We do this recursively:

SET ITER /IT1*IT5/;

PS.M = 1;
LOOP(ITER$(ABS(PS.M) GT 1.0E-8),
$INCLUDE CHKCAL.GEN
    SOLVE CHKCAL USING MCP;
    S.L = S.L - PS.M;
);
DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL. ", ETA;
ETA = ((S.L - S0) / 0.001) * (1 / S0);
DISPLAY ETA;

4.54 Glossary

<table>
<thead>
<tr>
<th>acronym</th>
<th>A GAMS data type used for storing non-numeric (aka logical or categorical) data. For more details, see section Acronyms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>An alternative name for a set, used to have multiple indices running over a common set. For more information, see section The Alias Statement: Multiple Names for a Set.</td>
</tr>
<tr>
<td>algorithm</td>
<td>A sequence of actions to perform in order to solve a problem. GAMS solvers contain algorithm implementations, so the two terms are sometimes used interchangeably.</td>
</tr>
<tr>
<td>assignment</td>
<td>The statement used to assign values to an identifier. For a detailed introduction, see section The Assignment Statement.</td>
</tr>
<tr>
<td>basic</td>
<td>A column is basic if it is in the basis maintained by the solution method for the problem in question (e.g. by the simplex method for LP). A row is basic if the elementary column associated with its slack variable is basic. For more details, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td>binding</td>
<td>An inequality constraint or variable bound is binding when the value of the associated slack is zero, i.e. when it is satisfied as an equality. See also Mathematical Programming Glossary.</td>
</tr>
<tr>
<td>bounds</td>
<td>Upper and lower limits on the possible values that a column may assume in a feasible solution. May be infinite, i.e. no limit is imposed.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>column</td>
<td>An individual decision variable in the model passed to a solver. Also called a single variable in the GAMS listing file. An indexed GAMS variable typically contains many columns.</td>
</tr>
<tr>
<td>compilation</td>
<td>The initial phase of GAMS processing, when the program is being checked for syntax and consistency.</td>
</tr>
<tr>
<td>compile-time constant</td>
<td>A compile-time constant is a string that will be replaced at compile-time with a fixed value. These constants are usually related to a function, model attribute or option and are used for their mnemonic and descriptive value. See section Compile-Time Constants.</td>
</tr>
<tr>
<td>compile-time variable</td>
<td>Special variables that are substituted with their values at compile-time. For further information, see section Compile-Time Variables.</td>
</tr>
<tr>
<td>constraint</td>
<td>A relationship between variables that must hold in a feasible solution. A constraint can refer to a single relationship (i.e. a row) or to a collection of rows, such as an equation. For further details, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td>continuous</td>
<td>Used to describe functions (in the usual mathematical way) and variables. Roughly speaking, a function is continuous if its graph has no breaks or jumps in it. A continuous variable may assume any value within its bounds, in contrast to binary or integer variables.</td>
</tr>
<tr>
<td>controlling sets</td>
<td>See driving sets.</td>
</tr>
<tr>
<td>data types</td>
<td>Each symbol or identifier must be declared as one of the available data types, e.g set, parameter, equation, or model. Note that scalars and tables are not separate data types, but convenient input formats for certain parameters. See also section Data Types and Definitions.</td>
</tr>
<tr>
<td>decision variable</td>
<td>A decision variable (aka endogenous variable) represents a decision to be made, e.g. the amount to produce or consume or the unit price to charge. Decision variables are the unknowns of a mathematical programming model. In contrast, exogenous variables or parameters are outside of the decision maker's control. See also the Mathematical Programming Glossary.</td>
</tr>
<tr>
<td>declaration</td>
<td>The introduction of an identifier along with the specification of its data type. A declaration may also include the specification of data or initial values, in which case it also acts as a definition. See also section Classification of GAMS Statements.</td>
</tr>
<tr>
<td>default</td>
<td>The value used, or the action taken, if the user provides no information.</td>
</tr>
<tr>
<td>definition</td>
<td>A statement specifying the content or data associated with a GAMS identifier, e.g. the equations in a model, the algebra of an equation, the elements of a set, or the initial values of a parameter. Definitions are processed during compilation. See also section Classification of GAMS Statements.</td>
</tr>
<tr>
<td>direction</td>
<td>The direction of optimization, i.e. maximization or minimization.</td>
</tr>
<tr>
<td>discontinuous</td>
<td>A function is discontinuous if it is not continuous. Sometimes, we call a function discontinuous if its derivatives are not continuous, i.e. it is nonsmooth.</td>
</tr>
<tr>
<td><strong>discrete</strong></td>
<td>A discrete variable is any variable that is not continuous. For example, binary and integer variables are discrete. For more details, see section &quot;Types of Discrete Variables.&quot;</td>
</tr>
<tr>
<td><strong>dollar control option</strong></td>
<td>Directives or options used to control input or output details associated with the GAMS compiler. They are introduced in chapter &quot;Dollar Control Options.&quot;</td>
</tr>
<tr>
<td><strong>dollar operator</strong></td>
<td>An operator used to indicate and define conditional expressions in assignment statements and equation definitions. For more information, see section &quot;The Dollar Condition.&quot;</td>
</tr>
<tr>
<td><strong>domain checking</strong></td>
<td>A system of checks that ensures that operations and relationships involving sets and domains are logically consistent. For example, referencing ( \text{pop}(t,\text{city}) ) when we have a declaration of ( \text{pop}(\text{city},t) ) is a logical error that domain checking will catch. For more information, see section &quot;Domain Checking.&quot;</td>
</tr>
<tr>
<td><strong>domain of definition</strong></td>
<td>The set of tuples (i.e. label combinations) for which an indexed equation is defined. When a solve statement is executed, a row of the equation is generated for each member of the domain of definition.</td>
</tr>
<tr>
<td><strong>domain restriction condition</strong></td>
<td>The narrowing of the domain of definition with a dollar operator. When used in this way, the dollar operator occurs on the left of the symbol &quot;.&quot; in an equation definition. For more details, see section &quot;Dollar Control over the Domain of Definition.&quot;</td>
</tr>
<tr>
<td><strong>driving set</strong></td>
<td>The set or list of sets that an indexed operation or domain runs or is defined over. Loops, indexed operations like ( \text{sum} ), and domains of definition all make use of driving sets.</td>
</tr>
<tr>
<td><strong>dynamic set</strong></td>
<td>A set is dynamic if it has been changed with an assignment statement. Dynamic sets cannot be used with lag operations or in domain declarations. For more information, see chapter &quot;Dynamic Sets.&quot;</td>
</tr>
<tr>
<td><strong>echo print</strong></td>
<td>In the output file, the echo print is a listing of the input with added line numbers. Details are given in section &quot;The Echo Print of the Input File.&quot;</td>
</tr>
<tr>
<td><strong>endogenous</strong></td>
<td>Endogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with variables, i.e. values that change when a solve statement is processed. See also the discussion in section &quot;Functions in Equation Definitions.&quot;</td>
</tr>
<tr>
<td><strong>equation</strong></td>
<td>The GAMS data type used to specify required relationships between activity levels of variables. They are introduced and discussed in detail in chapter &quot;Equations.&quot;</td>
</tr>
<tr>
<td><strong>execution</strong></td>
<td>The second phase of GAMS processing, when GAMS is actually carrying out data transformations or generating a model.</td>
</tr>
<tr>
<td><strong>execution statements</strong></td>
<td>Instructions to carry out actions such as data transformations, model solves and report generation. Examples include the assignment statement, the loop statement and the solve statement. For a full list, see section &quot;Classification of GAMS Statements.&quot;</td>
</tr>
<tr>
<td><strong>Exogenous</strong></td>
<td>Exogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with constants, i.e. values that do not change when a solve statement is processed. These are most often parameters but any variable or equation field will be treated exogenously when it appears in an equation. See also the discussion in section Functions in Equation Definitions.</td>
</tr>
<tr>
<td><strong>Explanatory text</strong></td>
<td>See text.</td>
</tr>
<tr>
<td><strong>Extended arithmetic</strong></td>
<td>In GAMS, the usual computer arithmetic is extended to include special values (e.g. inf) and the results of operations and functions that use them. For example, 6 + inf is inf and min(6, inf) is 6. See section Extended Range Arithmetic and Error Handling for details.</td>
</tr>
<tr>
<td><strong>External equation</strong></td>
<td>An equation defined in an external module. For example, on Windows systems, external equations are defined by a DLL. See section External Equations for details.</td>
</tr>
<tr>
<td><strong>Extrinsic function</strong></td>
<td>A function that is imported into a GAMS program from an external function library. Once imported, extrinsic functions can be used in the same way as intrinsic functions like cos and exp. See section Extrinsic Functions for details.</td>
</tr>
<tr>
<td><strong>E-format</strong></td>
<td>A convenient text-based (i.e. unsuperscripted) way to represent numbers in scientific notation in which the exponent of the 10 is prefixed by the letter e. For example, one US mile = 1.609344e+03 meters.</td>
</tr>
<tr>
<td><strong>Feasible</strong></td>
<td>Often used to describe a model (or a subset of constraints within a model) that has at least one feasible solution, but also used to describe a point that satisfies a set of constraints. For more information, see Mathematical Programming Glossary. See also infeasible.</td>
</tr>
<tr>
<td><strong>Feasible solution</strong></td>
<td>A solution to a model in which all column activity levels are within the bounds and all the constraints are satisfied. For more information, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>Identifier</strong></td>
<td>The name given to a data entity in a GAMS program. Also called a symbol. See section Identifiers for further details.</td>
</tr>
<tr>
<td><strong>Indexed operation</strong></td>
<td>An operation (e.g. sum or max) that is performed over one or more indices. See also section Indexed Operations.</td>
</tr>
<tr>
<td><strong>Inequality constraint</strong></td>
<td>A constraint in which the imposed relationship between the columns is not specified with an equality but instead with an inequality (e.g. “greater than or equal to”, “less than or equal to”). The GAMS syntax =G= and =L= is used in equation definitions to specify these relationships.</td>
</tr>
<tr>
<td><strong>Infeasible</strong></td>
<td>Not feasible. Used to describe either a model that has no feasible solution or an intermediate solution or point that is not feasible (although feasible solutions may exist).</td>
</tr>
<tr>
<td><strong>initialization</strong></td>
<td>Associating or assigning <em>initial values</em> to an identifier as part of the declaration or definition of the identifier. We typically talk about the initialization of only those identifiers (e.g. sets and parameters) that can be <em>assigned</em> different values later. For identifiers like models and equations, we say they are <em>defined</em>, not <em>initialized</em>.</td>
</tr>
<tr>
<td><strong>intrinsic function</strong></td>
<td>Functions that are provided by and part of GAMS, including mathematical, string, logical, time/calendar, and utility functions. Contrast to extrinsic functions.</td>
</tr>
<tr>
<td><strong>label</strong></td>
<td>Sets are built up from labels (aka <em>elements</em>). One-dimensional sets are collections of labels, while multi-dimensional sets contain tuples of labels. See section Labels for details.</td>
</tr>
<tr>
<td><strong>list</strong></td>
<td>See list format.</td>
</tr>
<tr>
<td><strong>list format</strong></td>
<td>GAMS data may be initialized by data in list format, i.e. a list where each tuple of labels in the data is specified in full. Data may also be displayed in list format, in which case each nondefault value is displayed via a fully-specified tuple of labels.</td>
</tr>
<tr>
<td><strong>macro</strong></td>
<td>A macro is a fragment of code that has been given a name: whenever the name is used, that name is replaced by the contents of the macro (i.e. the fragment of code). This is useful for defining and automating structured text replacement, e.g. to replace the text $MYOBJ(x)$ with $[sqr(x) + exp(x-2)/7]$. See section Macros in GAMS for details.</td>
</tr>
<tr>
<td><strong>marginal</strong></td>
<td>Marginal values (aka &quot;dual values&quot;, &quot;reduced costs&quot;, &quot;shadow prices&quot;, or &quot;multipliers&quot;) are stored in the &quot;<em>m</em>&quot; variable attribute or equation attribute. The GAMS sign convention is this: the marginal value represents the amount and direction of change in the objective value given a <em>unit increase</em> in the binding constant (e.g. an active variable bound or right-hand side). For further information, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>matrix element</strong></td>
<td>See nonzero element.</td>
</tr>
<tr>
<td><strong>model generation</strong></td>
<td>An initial step in processing a solve statement, where a model instance is generated (based on the equation definitions and the data referenced by those definitions) for the solver.</td>
</tr>
<tr>
<td><strong>model list</strong></td>
<td>The list of <em>equations</em> that are part of the model, as specified in the model statement. For further details, see section The Model Statement.</td>
</tr>
<tr>
<td><strong>model status</strong></td>
<td>An integer returned by the solver that gives information about the model (e.g. INFEASIBLE), about the point returned by the solver (e.g. LOCALLY INFEASIBLE), or both (UNBOUNDED NO SOLUTION RETURNED). For an overview of all values, see section Model Status.</td>
</tr>
<tr>
<td><strong>nonbasic</strong></td>
<td>A column that is not basic and (in nonlinear problems) not superbasic. Typically, nonbasic columns behave or are treated as if they are fixed at a bound. If the solution is feasible, the value of a nonbasic column will equal a finite bound (or zero if there are no finite bounds).</td>
</tr>
<tr>
<td><strong>nonlinear nonzero</strong></td>
<td>In a linear programming problem, the nonzero elements of the constraint matrix are constant. In a nonlinear problem, elements of the Jacobian matrix vary where variables appear nonlinearly. These non-constant Jacobian elements are called nonlinear nonzeros.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>nonoptimal</strong></td>
<td>Not optimal. A solution is nonoptimal if other solutions exist with better objective values. A variable is nonoptimal if its marginal (aka reduced cost) has the wrong sign. For example, in a minimization, a variable at lower bound having a negative marginal is nonoptimal. In the simplex method, a variable is nonoptimal if it is nonbasic and would improve the objective if it entered the basis.</td>
</tr>
<tr>
<td><strong>nonsmooth</strong></td>
<td>A function is nonsmooth if it is not smooth, i.e. if the function itself or its derivatives are not continuous. For example, the absolute value function is nonsmooth because of the kink at the origin: its derivative is not continuous there.</td>
</tr>
<tr>
<td><strong>nonzero element</strong></td>
<td>An element or coefficient of a matrix (e.g. the constraint matrix of an LP or the Jacobian matrix of an NLP) that is not zero. Most mathematical programming problems are sparse, i.e. only a small proportion of the elements in the constraint or Jacobian matrix are nonzero.</td>
</tr>
<tr>
<td><strong>objective row (or function)</strong></td>
<td>Solver systems often require the specification of a row or (for nonlinear systems) a function whose value will be optimized. A GAMS model, in contrast, is solved by specifying a scalar objective variable to be optimized.</td>
</tr>
<tr>
<td><strong>objective value</strong></td>
<td>The current value of the objective row or of the objective variable.</td>
</tr>
<tr>
<td><strong>objective variable</strong></td>
<td>The variable to be optimized, as specified in the solve statement.</td>
</tr>
<tr>
<td><strong>optimal</strong></td>
<td>A feasible solution in which the objective value is the best possible. For more details, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>optimality gap</strong></td>
<td>The optimality gap is a metric for the distance (or an upper bound on the distance) between the objective value of the current feasible solution and the optimal objective value. For more information, see Mathematical Programming Glossary. In GAMS, MIP and global solvers terminate when the optimality gap is sufficiently reduced: see the options optCA and optCR for details.</td>
</tr>
<tr>
<td><strong>option</strong></td>
<td>A control that allows users to change or influence the behavior or parameters in many different parts of the system. Options in GAMS may be set in three different ways: with model attributes, command line parameters, and option statements. A full list of all options with detailed descriptions is given in section Detailed Descriptions of All Options.</td>
</tr>
<tr>
<td><strong>ordered set</strong></td>
<td>A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the labels in the universe. Only sets that are ordered (and static) can be treated as sequences, i.e. used in lags and leads and with Ord-type operations. For more information, see chapter Sets as Sequences: Ordered Sets.</td>
</tr>
<tr>
<td><strong>output file</strong></td>
<td>A file, also called the listing file, produced (by default) by the run of a GAMS program. It contains output that describes or logs the run in question. For details see chapter GAMS Output.</td>
</tr>
<tr>
<td><strong>parameter</strong></td>
<td>A data type in GAMS used to store (indexed) constant data. For details on parameters, see chapter Data Entry: Parameters, Scalars and Tables.</td>
</tr>
<tr>
<td><strong>problem type</strong></td>
<td>A class of models that is characterized or defined by the type of algebra accepted (e.g. linear or nonlinear), the GLOSSARY_variable_type &quot;variable types&quot; allowed (e.g. continuous vs. discrete), and the definition of what it means to solve the model or problem (e.g. optimization vs. MCP). A list of all problem types in GAMS is given in section Classification of Models.</td>
</tr>
<tr>
<td><strong>program</strong></td>
<td>A GAMS input file. Typically a program defines a model and solves it, but programs can also work only with data or act as scripts that call other programs.</td>
</tr>
<tr>
<td><strong>relational operator</strong></td>
<td>This term may be used in two ways. First, in an equation definition, it describes the type of relationship that the equation specifies. An example is equality, as specified with the symbol $=$. For more information, see Table Equation Types. Second, in a logical expression, a relational operator compares two numerical expressions and returns a logical value. For details see section Logical Conditions: Numerical Relational Operators.</td>
</tr>
<tr>
<td><strong>right-hand side</strong></td>
<td>The value of the constant term in a constraint.</td>
</tr>
<tr>
<td><strong>row</strong></td>
<td>An individual constraint in the model passed to a solver. Also called a single equation in the GAMS listing file. An indexed GAMS equation typically contains many rows.</td>
</tr>
<tr>
<td><strong>scalar</strong></td>
<td>An un-indexed parameter, or the statement used to declare or define this parameter. See also section Scalars.</td>
</tr>
<tr>
<td><strong>set</strong></td>
<td>A collection of elements (aka labels) or element tuples. The set statement is used to declare and define a set. Sets are introduced and discussed in detail in chapter Set Definition.</td>
</tr>
<tr>
<td><strong>simplex method</strong></td>
<td>An algorithm often used to solve linear programming problems. For more details, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>singleton set</strong></td>
<td>A special set that has at most one element (zero elements are allowed as well). For more information, see section Singleton Sets.</td>
</tr>
<tr>
<td><strong>slack</strong></td>
<td>The amount by which an inequality constraint or variable bound is not binding.</td>
</tr>
<tr>
<td><strong>slack variable</strong></td>
<td>A non-negative variable introduced to represent the slack in an inequality constraint and to convert the inequality into an equality. For further information, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>smooth</strong></td>
<td>A function that is continuous and whose derivatives are all continuous is smooth. Sometimes we consider a function smooth if enough of its derivatives are continuous.</td>
</tr>
<tr>
<td><strong>solver</strong></td>
<td>An implementation of an algorithm or algorithms for solving models of a given problem type or types. An example is MINOS and QUADMINOS, which is used to solve both linear and nonlinear programming problems.</td>
</tr>
<tr>
<td><strong>solver status</strong></td>
<td>An integer returned by the solver that indicates the solver termination condition, i.e. why the solver stopped. For details, see section Solver Status.</td>
</tr>
<tr>
<td><strong>statements</strong></td>
<td>Statements (aka units or sentences) are the fundamental building blocks of GAMS programs: each program is a sequence of statements. Statements are used to declare identifiers, define equations, create loops, and solve models. A full list of GAMS statements is given in section Classification of GAMS Statements.</td>
</tr>
<tr>
<td><strong>static set</strong></td>
<td>Used in two slightly different ways: a set is static if it has not changed (i.e. it is not dynamic) or if it cannot change, i.e. it is immutable. Usually, sets are either used in ways that make them immutable (e.g. as domains or with lags and leads) or in ways that make them dynamic (i.e. by assigning to them): in such cases, static is synonymous with immutable.</td>
</tr>
<tr>
<td><strong>superbasic</strong></td>
<td>Some algorithms for nonlinear programming make use of superbasic variables that are neither basic (i.e. in the basis) nor nonbasic (i.e. temporarily fixed at a bound). These algorithms often search in the space defined by these superbasic variables. For further information, see Mathematical Programming Glossary.</td>
</tr>
<tr>
<td><strong>symbol</strong></td>
<td>An identifier.</td>
</tr>
<tr>
<td><strong>table</strong></td>
<td>A table statement is often a convenient way to define a parameter having two or more dimensions. For details see section Tables.</td>
</tr>
<tr>
<td><strong>text</strong></td>
<td>An optional description associated with an identifier or with an element of a set. For details see section Text.</td>
</tr>
<tr>
<td><strong>unique element</strong></td>
<td>A label used to define set membership.</td>
</tr>
<tr>
<td><strong>universe</strong></td>
<td>Also called the universal set, it is denoted by the symbol ‘*’ and contains all the labels that have been declared in the program plus any labels used as if they were declared (e.g. link(‘sink’,j) = NO;). If an identifier is declared using the universe, that effectively turns off or limits domain checking for that identifier. For more details, see section The Universal Set: * as Set Identifier.</td>
</tr>
<tr>
<td><strong>variable type</strong></td>
<td>Variables are classified or typed based on their default bounds and the values they are allowed to take. This latter classification partitions variables into continuous or discrete. For example, free variables have no default bounds and can take any value between their bounds: they are continuous. In contrast, binary variables can take only the values zero and one. See section Variable Types for details.</td>
</tr>
<tr>
<td><strong>zero default</strong></td>
<td>Some GAMS data (e.g. parameters, lower bounds for positive variables) have a default value of zero. These default zero values are not stored. However, unless other values are provided via initialization or assignment, zero will be used when the data value is taken.</td>
</tr>
</tbody>
</table>
Chapter 5

Solver Manuals

A large number of solvers for mathematical programming models have been hooked up to GAMS. The tables below provide a brief description of each solver, the model types each solver is capable of solving, and the platforms supported by each solver. For general information on using GAMS solvers, see Solver Usage.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Vendor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAEC 2.11</td>
<td>Abo University</td>
<td>MINLP solver based on the extended cutting plane (ECP) method</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>Princeton University</td>
<td>Deterministic global optimization for MINLP</td>
</tr>
<tr>
<td>BARON</td>
<td>The Optimization Firm, LLC</td>
<td>Branch-And-Reduce Optimization Navigator for proven global solutions</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>COIN-OR Foundation</td>
<td>High-performance LP/MIP solver</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>ARKI Consulting and Development</td>
<td>Large scale NLP solver</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>ARKI Consulting and Development</td>
<td>Large scale NLP solver</td>
</tr>
<tr>
<td>CONVERT</td>
<td>GAMS Development Corp</td>
<td>Framework for translating modes into scalar models of other languages</td>
</tr>
<tr>
<td>COPT 5.0</td>
<td>Cardinal Operations</td>
<td>High-performance LP/MIP solver</td>
</tr>
<tr>
<td>CPLEX 22.1</td>
<td>IBM ILOG</td>
<td>High-performance LP/MIP solver</td>
</tr>
<tr>
<td>DE</td>
<td>GAMS Development Corp</td>
<td>Generates and solves the deterministic equivalent of a stochastic program, included in EMP/SP</td>
</tr>
<tr>
<td>DECIS</td>
<td>G. Infanger, Inc.</td>
<td>Large scale stochastic programming solver</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>EDRC, Carnegie Mellon University</td>
<td>Framework for solving MINLP models</td>
</tr>
<tr>
<td>EXAMINER</td>
<td>GAMS Development Corp</td>
<td>A tool for examining solution points and assessing their merit</td>
</tr>
<tr>
<td>GAMSCHK</td>
<td>Bruce McCarl</td>
<td>A System for Examining the Structure and Solution Properties of Linear Programming Problems Solved using GAMS</td>
</tr>
<tr>
<td>GUROBI 9.5</td>
<td>Gurobi Optimization</td>
<td>High performance LP/MIP solver</td>
</tr>
<tr>
<td>GUSS</td>
<td>GAMS Development Corp</td>
<td>A framework for solving many instances of related models efficiently (Gather-Update-Solver-Scatter)</td>
</tr>
<tr>
<td>HiGHS 1.2</td>
<td>ERGO</td>
<td>High performance LP/MIP solver</td>
</tr>
<tr>
<td>Solver</td>
<td>Vendor</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>COIN-OR Foundation</td>
<td>Interior Point Optimizer for large scale nonlinear programming</td>
</tr>
<tr>
<td>JAMS</td>
<td>GAMS Development Corp</td>
<td>Solver to reformulate extended mathematical programs (incl. LogMIP)</td>
</tr>
<tr>
<td>KESTREL</td>
<td>NEOS</td>
<td>Framework for using remote NEOS solvers with a local GAMS system</td>
</tr>
<tr>
<td>KNITRO 13.1</td>
<td>Artelys</td>
<td>Large scale NLP solver</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>Lindo Systems Inc.</td>
<td>A stochastic solver from Lindo Systems, Inc. Includes an unrestricted version of LINDOGLOBAL</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>Lindo Systems Inc.</td>
<td>MINLP solver for proven global solutions</td>
</tr>
<tr>
<td>MILES</td>
<td>University of Colorado at Boulder</td>
<td>Solver to reformulate extended mathematical programs (incl. LogMIP)</td>
</tr>
<tr>
<td>MINOS 5.6</td>
<td>Stanford University</td>
<td>NLP solver</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>MOSEK ApS</td>
<td>Large scale mixed-integer conic programming solver</td>
</tr>
<tr>
<td>NLPEC</td>
<td>GAMS Development Corp</td>
<td>MPEC to NLP translator that uses other GAMS NLP solvers</td>
</tr>
<tr>
<td>OCTERACT 4</td>
<td>Octeract</td>
<td>MINLP solver for proven global solutions</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>Optimization Direct Inc</td>
<td>ODHeuristic on top of Cplex</td>
</tr>
<tr>
<td>OsiCplex</td>
<td>COIN-OR Foundation</td>
<td>Bare-Bone link to CPLEX</td>
</tr>
<tr>
<td>OsiGurobi</td>
<td>COIN-OR Foundation</td>
<td>Bare-Bone link to Gurobi</td>
</tr>
<tr>
<td>OsiMosek</td>
<td>COIN-OR Foundation</td>
<td>Bare-Bone link to Mosek</td>
</tr>
<tr>
<td>OsiXpress</td>
<td>COIN-OR Foundation</td>
<td>Bare-Bone link to Xpress</td>
</tr>
<tr>
<td>PATHNLP</td>
<td>University of Wisconsin - Madison</td>
<td>Large scale NLP solver for convex problems</td>
</tr>
<tr>
<td>PATH</td>
<td>University of Wisconsin - Madison</td>
<td>Large scale MCP solver</td>
</tr>
<tr>
<td>QUADMINOS 5.6</td>
<td>Stanford University</td>
<td>Quad-precision NLP solver</td>
</tr>
<tr>
<td>SBB</td>
<td>ARKI Consulting and Development</td>
<td>Branch-and-Bound algorithm for solving MINLP models</td>
</tr>
<tr>
<td>SCIP 8.0</td>
<td>Zuse Institute Berlin et.al.</td>
<td>High-performance Constraint Integer Programming solver</td>
</tr>
<tr>
<td>SELKIE</td>
<td>University of Wisconsin - Madison</td>
<td>Decomposition and parallel solution for EMP</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>Abo Akademi University</td>
<td>MINLP solver based on the extended supporting hyperplane (ESH) method</td>
</tr>
<tr>
<td>SNOPT 7.7</td>
<td>Stanford University</td>
<td>Large scale SQP based NLP solver</td>
</tr>
<tr>
<td>SOPLEX 6.0</td>
<td>Zuse Institute Berlin</td>
<td>High-performance LP solver</td>
</tr>
<tr>
<td>XA</td>
<td>Sunset Software</td>
<td>Large scale LP/MIP solver</td>
</tr>
<tr>
<td>XPRESS 40.01</td>
<td>FICO</td>
<td>High performance LP/MIP and SLP based MINLP solver</td>
</tr>
</tbody>
</table>

### 5.1 Model Types

GAMS is able to formulate models in many different types of problem classes or model types. Typically, a solver will be capable of solving (i.e. will accept as input) more than one model type. The solver/model type matrix shows which solver is capable of which model type:
### 5.1 Model Types

<table>
<thead>
<tr>
<th>LP</th>
<th>MIP</th>
<th>NLP</th>
<th>MCP</th>
<th>MPEC</th>
<th>CNS</th>
<th>DNLI</th>
<th>MINI</th>
<th>QCP</th>
<th>MIQC</th>
<th>Stoch. Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHECnP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT/3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT/4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HiGHS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCTERACT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QUALMINOS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNOPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soplex</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XPRESS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

* Deterministic global solver

When choosing a solver, some judgement should be applied when considering the listed model type capabilities for the solver - the same capability "check boxes" does not imply equality in capacity or suitability. For example, take a hypothetical solver WeOpt designed to solve MINLP models. Since the problem class MINLP includes NLP, MIP, and LP as subclasses, solver WeOpt could include these capabilities also. If WeOpt is also a good performer on NLP models, it would include that capability. But if it does not shine at all as a MIP or LP solver, we would choose not to include MIP and LP in the capability list for WeOpt. In such a case one can always solve using a more general model type (e.g. solve an LP model as NLP so WeOpt can be used), but WeOpt will not advertise itself as an LP solver. Since the WeOpt solver does not even recognize MCP or MPEC models, we don't include those capabilities.

There are two types of non-linear solvers: local and global. A local solver can find a local optimum but generally cannot comment on global optimality of the solution. A local optimum is a point where the objective value is better than its neighboring points but could be worse than a distant point. On the other
hand, a global solver can find and prove that the obtained solution is a global optimum i.e., there is no feasible solution that would result in a better objective value. It is worth noting that a local solver can solve a model to global optimality in some special cases. In the table shown, the entries with * in the column Global indicate solvers that can solve a model to global optimality.

There are other differences in solvers that are difficult to quantify or cannot be captured by a capability table like the one shown. For example, for nonconvex NLP or QCP models, one solver could look only for first-order stationary points, another for local solutions, a third for local solutions using a scatter search or similar search heuristic, and a fourth could do a true global search for the global optimum. The relative merits (measured typically by speed alone) of solvers is the subject of considerable benchmarking activity and discussion.

The GAMS sales team can help answer questions you may have about solver capability. We also offer free evaluation licenses to help you decide what solvers are most suitable for your models.

### 5.2 Supported Platforms

The solver/platform matrix shows which platforms each solver is supported on. In addition, where a vendor has discontinued solver support for a particular platform and we continue to ship the last available supported version, this version number is indicated as well.

<table>
<thead>
<tr>
<th>Solver</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAEC 2.11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANTIGONE 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BARON</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBC 2.10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CONOPT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>COPT 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPLEX 22.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DECIS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DICOPT 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUROBI 9.5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUSS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IPOPT 3.14</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HiGHS 1.2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KESTREL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KNITRO 13.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDO 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LINDOGLOBAL 13.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MILES</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MOSEK 9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLPEC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCTERACT 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ODHCPLEX 6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PATH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QUADMINOS 5.6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCIP 8.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHOT 1.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Author
Tapio Westerlund (twesterl@abo.fi) and Toni Lastusilta. Åbo Akademi University, Finland
Ville-Pekka Eronen. University of Turku, Finland

5.3 AlphaECP

5.3.1 Introduction

AlphaECP is a Mixed-Integer Non-Linear Programming (MINLP) solver based on the extended cutting plane (ECP) method. The solver can be applied to general MINLP problems and it can ensure global optimal solutions for pseudo-convex MINLP problems. The documentation is written as if the considered problem is a minimization problem, if not otherwise denoted.

The ECP method is an extension of Kelley’s cutting plane method, which was originally given for convex NLP problems [105]. The method only requires the solution of a MIP sub-problem in each iteration. The MIP sub-problems can be solved to optimality, to feasibility, or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. Further information about the underlying algorithm can be found in [200] and [145] [174] [199] [201].

Further developments of the GAMS/AlphaECP algorithm have introduced additional functionality. A NLP solver can now be called at MIP solutions. This improves AlphaECP’s ability to find feasible and accurate solutions, especially for MINLP problems which mainly contain continuous variables. Furthermore, a heuristic that reselects cutting planes during the iteration procedure can be used to improve the capability of solving non-convex problems. In addition to the termination based on constraint tolerance the algorithm can terminate also if relative objective gap termination criterion is satisfied. This is only supported for convex problems.

5.3.1.1 Licensing and software requirements

Users need to have a GAMS/AlphaECP license in order to use GAMS/AlphaECP. In addition, a licensed MIP solver is required for solving the mixed integer sub-problem, and a licensed NLP solver is required if the NLP option is used.

5.3.1.2 Running AlphaECP

AlphaECP solves MINLP models. If AlphaECP is not specified as the default solver for these models it can be invoked by issuing the following command before the solve statement:

```
option minlp=alphaecp, miqcp=alphaecp;
```
In principle AlphaECP can also handle NLP models, but it is more suitable for MINLP problems. However, when combined with an NLP solver it can find solutions the NLP solver cannot find by itself. In this case it acts as a good starting point generator. If you want to solve NLPs with AlphaECP you need to \textit{trick} the GAMS system by solving your NLP as an MINLP:

\begin{verbatim}
solve mynlpmodel minimizing obj using minlp;
\end{verbatim}

Constraint violations are reported throughout the progress of AlphaECP and at the end of the algorithm. The violation is reported for the non-linear constraints only. The violation of the linear constraints is subject to the feasibility tolerance of the MIP/NLP solver.

### 5.3.2 GAMS/AlphaECP Output

The log output below is obtained for the MINLP model fuel.gms from the GAMS model library:

---

Welcome to AlphaECP v2.11.01
MINLP Problem Solver using the Extended Cutting Plane Approach.
Method development - T.Westerlund, Abo Akademi University, FIN
Algorithm development - T.Lastusilta, Abo Akademi University, FIN
Algorithm development - V.-P. Eronen, Turku University, FIN

Minimization problem: "fuel.gms"
The GAMS-model has in total 39 elements of which 15\% are non-linear(NL)
included in 16 constraints of which 25\% are NL
The NL constraint signs: =E=(3), =Q=(1), =L=(0)
The number of variables in NL elements are 6 from a total of 16
variables: Continuous(13), Binary(3), Integer(0)
---

Using following settings
AlphaECP option file optfile=0
Time limit for AlphaECP (in seconds) reslim=10000000000
SolveLink for NLP and MIP sub-solver solvelink=5
Solver trace file solvetrace=(Inactive)
Cutting plane strategy (0-3) CUTdelcrit=3
Cut generation pace CUTnrcuts=0
Updating multiplier if MIP is infeasible ECPbeta=1.3
Write encountered solutions to gdx files ECPdumpsol=0
Updating multiplier when verifying solution ECPgamma=2
Maximum number of AlphaECP iterations ECPiterlim=-1
Level of AlphaECP output to statusfile (0-4) ECPloglevel=0
Master strategy (0=Unknown 1=Convex) ECPmaster=0
Return solution (1=MIP/2=NLP/...) ECPretsol=2
User specified start-point (0-3) ECPstart=3
AlphaECP strategy (1-5) ECPstrategy=2
AlphaECP termination criterion (1-2) ECPtoltype=1
Upper limit of considered MIP solutions per MIP call MIPnrsols=50
Relative MIP gap in intermediate sub-problems (0->1.0) MIPoptcr=1.00
Initial iteration limit when MIPoptcr is reduced MIPoptcrlim=200
Strategy to increase MIPoptcrlim MIPoptcrlimtype=0
MIP is solved to optimality with this frequency MIPoptimaliter=0
Strategy for multiple MIP solutions MIPsolstrat=1
MIP solver for sub-problems and . option file number MIPsolver=cplex.0
---
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Stepcode, Number</th>
<th>Point Alpha</th>
<th>OPT</th>
<th>Movement</th>
<th>Violation</th>
<th>Maximum</th>
<th>MIObjval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start-point: NL constraint (1) infeasible</td>
<td>0</td>
<td>H 0 0 0 1 0 4</td>
<td>1.8E+03</td>
<td>NA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SAFGI 1 1 1 1 9.3E+03 0</td>
<td>1.1E-13</td>
<td>8566.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SAFH 1 1 0 1 6.6E+03 4</td>
<td>1.8E+03</td>
<td>4844.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SAFH 3 2 0 1 8.4E+03 3</td>
<td>1.8E+03</td>
<td>7031.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SAFH 4 3 0 1 1.1E+03 2</td>
<td>1.8E+03</td>
<td>10157</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SAH 5 4 0 1 0 1</td>
<td>7E+02</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SAH 6 5 0 1 1.1E+03 2</td>
<td>3.5E+02</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SAH 8 6 0 1 1.9E+03 2</td>
<td>5.3E+02</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SAH 9 7 0 1 8.8E+02 3</td>
<td>2.6E+02</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SAH 12 8 0 1 8.8E+02 3</td>
<td>1.3E+02</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SAH 15 9 0 1 4.4E+02 3</td>
<td>66</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>79</td>
<td>SAIJL 101 65 0 1 0 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>SAJ 100 64 0 0.5 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>SAJ 100 64 0 0.3 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>AI 106 39 1 0 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>AI 106 39 1 0 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>AIJ 106 39 0 0 0</td>
<td>0.00067</td>
<td>11925</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AlphaECP: Iteration procedure terminated normally

---

Problem : fuel.gms
Solver Status : Normal Completion
Model Status : Locally Optimal
Exit comment : No Issues
Final solution : NLP
Objective value : 8566.1189616876672517
Max constraint (4) : 1.1368683772161602974e-13
Alternative solution : MIP
Alt. objective value : 8566.1189616876672517
Max constraint (4) : 1.1368683772161602974e-13
Time used (seconds) : 0.81
Time limit (seconds) : 10000000000
Iterations used : 102
Iteration limit : -1
Function evaluations : 496
Gradient evaluations : 186
Domain violations : 0
Gradients unusable : 0
Alphamax bound violations : 0
ECP time usage : 3.9 %
NLP time usage : 3.8 %
MIP time usage : 92.3 %
Optimal/total MIPs : 19/102
NLP solver calls    : 8
--------------------------------------------------------------------------

In every iteration, information about the MIP problem and modifications to it is given in 10 columns. Here is a description of the different columns:

**Iteration:** Iteration identifier.

**Stepcode, Problems:** Letter for what actions were taken in this iteration, i.e. MIP problem modifications before the next iteration.

- **A:** MIP solver feasible.
- **B:** MIP solver feasible after moving cutting planes, i.e. alpha update.
- **C:** MIP solver feasible after moving cutting planes close to their generation point. The movement is done to make it easier to satisfy nonlinear equality constraints.
- **D:** Line search was successful (in ECPstrategy 3).
- **E:** Line search failed (in ECPstrategy 3).
- **F:** A NLP solver was called.
- **G:** Found a MINLP solution.
- **H:** Added linearization(s) to the next MIP problem.
- **I:** Updated alpha values and possibly added linearizations.
- **J:** All cutting planes are valid underestimators for the pseudo-convex constraints, except for the nonlinear objective function constraint.
- **K:** The nonlinear objective function constraint value and MIP solution value differ more than $\epsilon_f$. A linearization was done to reduce the difference (in ECPstrategy 3).
- **L:** Removed all temporal linearizations.
- **M:** Domain violation(s), some of the constraint could not be evaluated.
- **N:** Some cutting plane(s) could not be generated because of gradient problems.
- **O:** No cutting planes could be generated.
- **P:** Reselecting cuts because cutting planes are repeatedly moved close to their generation point.
- **Q:** Added temporal linearization(s).
- **R:** Failed to add temporal linearization(s).
- **S:** MIP solver strategy to find encountered solutions selected.
- **T:** MIP solver strategy to require MIPnrsols solutions selected.
- **U:** MIP solver strategy to require MIPnrsols solutions with a MIPoptcr $\leq 0.2$ selected.
**Number of Cuts:** The number of cutting planes the solved MIP problem had.

**Point usage:** Number of points used to generate the cuts in the solved MIP problem.

**Alpha Upd.:** The number of times the alpha values has been increased.

**OPTCR:** Relative MIP gap. Note that this is different from TOLoptcr.

**Movement Norm:** The Euclidean norm of the current and previous MIP solution.

**Viol Cons:** Number of unsatisfied (violating) nonlinear constraints.

**Maximum Violation:** The most violating nonlinear constraint value.

**MIPobjval/NLobjval:** MIP or nonlinear objective variable value depending on ECPstrategy setting.

The cut reselection heuristic is called in the following cases:

1. If the MIP solver would otherwise return infeasible.
2. When the violation is not reducing, but the cutting planes are repeatedly moved close to their generation point.
3. When the violation is not reducing and domain violations are repeatedly encountered.

The heuristic reselects cutting planes in different ways, but always ensures that the same point cannot be found twice.

\[
\text{Pointusage} \quad 6/90 \quad \text{Cutusage} \quad 15/341 \quad (0,135)
\]

**Pointusage** informs how many points of all usable points have been used to generate the cutting planes. **Cutusage** tells how many cuts of all usable cuts have been used. The first number in \((0,135)\) tells how many cuts is required by the user, see CUTnrcuts and the second number gives the sum of added and removed cuts, i.e. a measure of how much the MIP problem has been modified. AlphaECP may fix some cuts and remove points and cuts during the cut reselection procedure in order to save memory.

At the end of each solve AlphaECP gives a summary which includes Problem, Solver Status, Model Status, etc. Note the following lines:

- **Exit comment** may give further information than solverstatus on why the solution procedure stopped.
- **Domain violations** (function evaluation failed) or **Gradients unusable** (all gradients < TOLgrad) might be caused by poor variable bounds.
- **Alphamax bound violations** inform the user how many times an alphamax value was calculated to be \(>10^{154}\) and was reset to \(10^{154}\).
5.3.3 Notes about Options

To instruct AlphaECP to read an option file you may use `ModelName.OptFile = 1`. The name of the option file is in this case `alphaecp.opt`. For further information, see The Solver Options File. AlphaECP supports the GAMS parameters `reslim` and `optCR`, however, other GAMS parameters are passed on to the sub-solvers. Note that `optCR` is transferred to option `TOoptcr`. Furthermore, you may also pass additional sub-solver specific options to the sub-solvers. For example, if you want to use all available threads and sub-solver CPLEX in opportunistic parallel search mode, then you may specify this in a GAMS model, in a similar way, as follows:

```gams
Model m / all /;
Option threads=0;
m.optfile=1;
$echo MIPsolver cplex.1 > alphaecp.opt
$echo parallelmode -1 > cplex.opt
Solve m using MINLP minimizing objvar;
```

The following information is worth noting when you are interested in AlphaECP options. A linearization of a nonlinear constraint is called a cutting plane or cut. Here a point refers to the variable levels. Global optimality can be guaranteed for pseudo-convex problems. However, if the objective variable is in a nonlinear constraint and pseudo-convex, then `ECPstrategy >= 3` needs to be used to guarantee global optimality (because one non-linear equality constraint makes a problem non-pseudo-convex, and hence also non-convex). The basic options might significantly impact the solution procedure, and the best values are likely to be problem specific. The user is therefore encouraged to try different values for the basic options.

5.3.4 Summary of AlphaECP Options

5.3.4.1 Basic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUTnrcuts</td>
<td>Cut generation pace</td>
<td>0</td>
</tr>
<tr>
<td>ECPmaster</td>
<td>Master strategy (0=User 1=Convex)</td>
<td>0</td>
</tr>
<tr>
<td>MIPnrsols</td>
<td>Upper limit of considered MIP solutions per MIP call</td>
<td>50</td>
</tr>
<tr>
<td>MIPsolstrat</td>
<td>MIP solution collection strategy</td>
<td>1</td>
</tr>
<tr>
<td>MIPsolver</td>
<td>MIP solver for sub-problems and . option file number</td>
<td>GAMS MIP solver</td>
</tr>
<tr>
<td>NLPsolver</td>
<td>NLP solver for sub-problems and . option file number</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>reslim</td>
<td>Time limit for AlphaECP (in seconds)</td>
<td>GAMS reslim</td>
</tr>
</tbody>
</table>

5.3.4.2 Algorithmic options for advanced users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUTdelcrit</td>
<td>Cutting plane strategy</td>
<td>3</td>
</tr>
<tr>
<td>ECPbeta</td>
<td>Updating multiplier if MIP is infeasible</td>
<td>1.3</td>
</tr>
<tr>
<td>ECPdumpsol</td>
<td>Write encountered solutions to gdx files</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.3.4.3 MIP Solver related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPloglevel</td>
<td>Level of MIP solver output</td>
<td>0</td>
</tr>
<tr>
<td>MIPoptcr</td>
<td>Relative MIP gap in intermediate sub-problems</td>
<td>1.0</td>
</tr>
<tr>
<td>MIPoptcrlim</td>
<td>Initial iteration limit when MIPoptcr is reduced</td>
<td>200</td>
</tr>
<tr>
<td>MIPoptcrlimtype</td>
<td>Strategy to increase MIPoptcrlim</td>
<td>0</td>
</tr>
<tr>
<td>MIPoptimaliter</td>
<td>MIP is solved to optimality with this frequency</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.3.4.4 NLP Solver related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLPcall</td>
<td>NLP strategy</td>
<td>5</td>
</tr>
<tr>
<td>NLPcalliter</td>
<td>NLP solver call at next (incremental) iteration</td>
<td>0</td>
</tr>
<tr>
<td>NLPcallsameint</td>
<td>NLP call after a number of recurring integer solutions</td>
<td>5</td>
</tr>
<tr>
<td>NLPloglevel</td>
<td>Level of NLP solver output</td>
<td>0</td>
</tr>
</tbody>
</table>
5.3.5 Detailed Descriptions of AlphaECP Options

**CUTdelcrit (integer):** Cutting plane strategy

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not remove any valid cuts.</td>
</tr>
<tr>
<td>1</td>
<td>As 0 and allow temporary cuts at semirandom points if normal cuts can not be made.</td>
</tr>
<tr>
<td>2</td>
<td>Allow temporary cuts and cut reselection, and use memory to save points and cuts.</td>
</tr>
<tr>
<td>3</td>
<td>As 2 and call the reselection heuristic before termination to improve the solution.</td>
</tr>
</tbody>
</table>

**CUTnrcuts (real):** Cut generation pace

The number of linearizations that are generated during an iteration can be chosen by AlphaECP, proportional to the number of violating constraints, or can be determined by a fixed amount. Furthermore, the cut reselection \( \text{CUTdelcrit} \geq 2 \) adds cuts to the problem so that the requested cut generation pace is taken into consideration.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let AlphaECP decide.</td>
</tr>
<tr>
<td>( 0 &lt; n &lt; 1 )</td>
<td>Number of linearizations = ( n \times ) the number of linearizations that is possible to generate.</td>
</tr>
<tr>
<td>&gt;1</td>
<td>Specifies the number of linearizations to generate.</td>
</tr>
</tbody>
</table>

**ECPbeta (real):** Updating multiplier if MIP is infeasible

In case of an infeasible MIP solution, the invalid cuts are updated with the \( \text{ECPbeta} \) multiplier.

Range: \([1.001, \infty]\)

Default: 1.3

**ECPdumpsol (integer):** Write encountered solutions to gdx files

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No.</td>
</tr>
<tr>
<td>1</td>
<td>Solutions that the NLP solver found.</td>
</tr>
<tr>
<td>2</td>
<td>Solutions that the NLP or MIP solver found.</td>
</tr>
</tbody>
</table>

**ECPgamma (real):** Updating multiplier when verifying solution

...
If a MINLP solution is obtained but some cuts are not valid underestimators they are updated with the ECPgamma multiplier in order to make them into valid underestimators.

Range: \([1.001, \infty]\)

Default: 2.0

**ECPiterlim (integer):** Maximum number of AlphaECP iterations

This is the maximum number of iterations given to AlphaECP to perform the optimization. Value -1 deactivates the AlphaECP iteration limit.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No limit.</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>Specifies an iteration limit.</td>
</tr>
</tbody>
</table>

**ECPloglevel (integer):** Level of AlphaECP output to statusfile

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No additional output to statusfile.</td>
</tr>
<tr>
<td>1</td>
<td>Report solutions. Report all encountered solutions with their corresponding variable levels.</td>
</tr>
<tr>
<td>2</td>
<td>Report main actions at iteration level (available for minimization problems).</td>
</tr>
<tr>
<td>3</td>
<td>Report main actions at linearization level (available for minimization problems).</td>
</tr>
<tr>
<td>4</td>
<td>Full reporting. Report the main actions taken, the linearizations, function values, and solution points for every iteration and line search details (available for minimization problems).</td>
</tr>
</tbody>
</table>

**ECPmaster (integer):** Master strategy (0=User 1=Convex)

The master strategy sets some options in order to solve a model with specific characteristics more efficiently. The affected options are noted in the log output. The set options take precedence over the value set by the user for the affected options.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use only user defined options.</td>
</tr>
<tr>
<td>1</td>
<td>The model is convex. Set option ECPstrategy, CUTdelcrit and ECPtoltype.</td>
</tr>
</tbody>
</table>

**ECPpcostrategy (integer):** Pseudo-convex objective function strategy

Default: 3
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Remove support. Remove old support planes when a new pseudo-convex problem is formed.</td>
</tr>
<tr>
<td>2</td>
<td>Replace support. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed.</td>
</tr>
<tr>
<td>3</td>
<td>Remove support and line search. Remove old support planes when a new pseudo-convex problem is formed and perform a line search when it is possible.</td>
</tr>
<tr>
<td>4</td>
<td>Replace support and line search. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed and perform a line search when it is possible.</td>
</tr>
</tbody>
</table>

**ECPretsol (integer):** Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE) ←

The reported solution can be extracted from either the MIP or NLP solver result. If the MIP solution is returned only the primal values are available.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Choose MIP solution if it is available.</td>
</tr>
<tr>
<td>2</td>
<td>Choose NLP solution if it is available.</td>
</tr>
<tr>
<td>3</td>
<td>Choose the solution with the best tolerance.</td>
</tr>
<tr>
<td>4</td>
<td>Choose the solution with the best objective value.</td>
</tr>
</tbody>
</table>

**ECPstart (integer):** User specified start-point ←

Define which variable levels are used when the optimization is started.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use a start-point; start the algorithm by solving the linear part (MIP) of the problem.</td>
</tr>
<tr>
<td>1</td>
<td>Use the user specified startpoint, but adjust the variable levels with a small value.</td>
</tr>
<tr>
<td>2</td>
<td>Use the exact start-point set by the user.</td>
</tr>
<tr>
<td>3</td>
<td>Use the exact start-point if linearly feasible; else adjust variable levels with a small value.</td>
</tr>
</tbody>
</table>

**ECPstrategy (integer):** AlphaECP strategy ←

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Convex strategy. Ensures global optimality for problems with convex objective function and convex constraints.</td>
</tr>
<tr>
<td>2</td>
<td>Pseudo-convex constraints. Ensures global optimality for problems with convex objective function and pseudo-convex constraints.</td>
</tr>
</tbody>
</table>
Pseudo-convex objective. Ensures global optimality for problems with pseudo-convex objective function and pseudo-convex constraints. The reformulation of a non-linear objective function into a constraint must be done in a specific way. The requirement is that the objective variable must be in a linear part of the non-linear function. Assuming that the minimized or maximized variable is called objvar, the reformulation can be done as follows: (objective function expression) - objvar =E= 0. Furthermore, this strategy can effectively use a feasible start-point.

4 Pseudo-convex objective, but first complete with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

5 Pseudo-convex objective, but find the first solution with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

ECPtoltpe (integer): AlphaECP termination criterion

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use termination based on tolerances TOLeps and TOLepsf.</td>
</tr>
<tr>
<td>2</td>
<td>Terminate also if relative objective gap is satisfied according to TOLopter. The relative objective gap termination criterion also called relative termination criterion is only supported for convex problems when using ECPstrategy=1. The relative termination criterion can be used for non-convex problems, as well as, with ECPstrategy=2, however, in this case it may not work properly because the lower bound may be invalid.</td>
</tr>
</tbody>
</table>

MIPloglevel (boolean): Level of MIP solver output

By default the detailed log of the MIP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS LogOption is set to 1 or 3, the MIP log will be merged into the AlphaECP log.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output.</td>
</tr>
<tr>
<td>1</td>
<td>MIP solver log goes to GAMS log.</td>
</tr>
</tbody>
</table>

MIPnrscs (integer): Upper limit of considered MIP solutions per MIP call

When the MIP solver returns several solutions the most suitable solution is chosen. Many times the solutions from the MIP solver are similar and a larger number might help to find a feasible MINLP solution if the constraints are almost satisfied. See MIPsolstrat to change the solution collection strategy.

Range: {1, ..., ∞}

Default: 50

MIPoptcr (real): Relative MIP gap in intermediate sub-problems
The relative stopping tolerance sent to the MIP solver for intermediate MIP problems. Note that the `MIPoptcr` value is decreased automatically to zero during the optimization.

**Range:** $[0, 1]$

**Default:** 1.0

**MIPoptcrlim (integer):** Initial iteration limit when MIPoptcr is reduced

The `MIPoptcr` parameter is reduced in steps: From 1 to 0.5 to 0.3 to 0.2 to 0.1 to 0.0. The first reduction is at iteration `MIPoptcrlim`. The following reductions occur also at iteration `MIPoptcrlim` but it is updated with a strategy defined by parameter `MIPoptcrlimtype`. Note that a step reduction may happen for other reasons.

**Range:** $\{1, \ldots, \infty\}$

**Default:** 200

**MIPoptcrlimtype (integer):** Strategy to increase MIPoptcrlim

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Exponential increase of MIPoptcrlim. The parameter MIPoptcrlim is increased by multiplying it by 2. The increase occurs at iteration &quot;MIPoptcrlim&quot;. If MIPoptcrlim=200 it will have values 200,400,800 etc.</td>
</tr>
<tr>
<td>1</td>
<td>Linear increase of MIPoptcrlim. The parameter MIPoptcrlim is increased by adding the original value of MIPoptcrlim to it. If MIPoptcrlim=200 it will have values 200,400,600 etc.</td>
</tr>
</tbody>
</table>

**MIPoptimaliter (integer):** MIP is solved to optimality with this frequency

Defines the frequency to solve the MIP problem to optimum before the algorithm sets `MIPoptcr=0` and, hence, solves MIP problems to optimum. For example if `MIPoptimaliter=n`, then at least every nth iteration the MIP problem is solved to the optimum. Solving MIP problem to optimum yields a lower bound for a convex MINLP problem which can help to terminate the algorithm faster if `ECPtoltype=2` is used. Non-default value is not recommended for non-convex problems nor if `ECPstrategy >1`.

**Default:** 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never.</td>
</tr>
<tr>
<td>n&gt;0</td>
<td>Every nth iteration, where n is the given value.</td>
</tr>
</tbody>
</table>

**MIPsolstrat (integer):** MIP solution collection strategy

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Instruct MIP solver to return only one solution.</td>
</tr>
<tr>
<td>1</td>
<td>Instruct MIP solver to return any solutions encountered during MIP procedure.</td>
</tr>
</tbody>
</table>
### MIPsolver (string): MIP solver for sub-problems and . option file number

MIPsolver specifies the GAMS MIP solver for sub-problems and the option file number. The solver is the name of the GAMS MIP solver and `n` is the integer corresponding to `optfile`. The option file is appended to the option file that is written by AlphaECP. Hence, the specified options take precedence over the options set by AlphaECP. If the `.n` is missing, the `optfile` is treated as zero; i.e., the MIP solver will not look for an options file specified by the user. This option can be used to overwrite the default that uses the MIP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: GAMS MIP solver

### NLPcall (integer): NLP strategy

NLPcall determines when the NLP solver is called.

Default: 5

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output.</td>
</tr>
<tr>
<td>1</td>
<td>Call the NLP solver at end of AlphaECP algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>As 1 and when a better solution is found.</td>
</tr>
<tr>
<td>3</td>
<td>As 2 and when the same integer solution is encountered NLPlimsameint times.</td>
</tr>
<tr>
<td>4</td>
<td>Let AlphaECP decide.</td>
</tr>
<tr>
<td>5</td>
<td>Let AlphaECP decide and add noise to the variable levels before call.</td>
</tr>
</tbody>
</table>

### NLPcalliter (integer): NLP solver call at next (incremental) iteration

NLPcalliter specifies an iteration interval for the NLP solver calls.

Default: 0

### NLPlimsameint (integer): NLP call after a number of recurring integer solutions

NLPlimsameint specifies the number of recurring integer solutions before the NLP solver is called. If the same integer solution is encountered `NLPlimsameint` times in a row then the NLP solver is called. The counter is reset after the NLP solver is called.

Range: `{1, ..., ∞}`

Default: 5

### NLPloglevel (boolean): Level of NLP solver output

NLPloglevel specifies the level of NLP solver output. By default, the detailed log of the NLP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS `LogOption` is set to 1 or 3, the NLP log will be merged into the AlphaECP log.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output.</td>
</tr>
<tr>
<td>1</td>
<td>NLP solver log goes to GAMS log.</td>
</tr>
</tbody>
</table>

**NLPreslim** *(real)*: NLP time limit per call

The time limit (in seconds) given to the chosen NLP solver at each NLP solver call. Setting this option to 0 calculates a time limit which is relative to the problem size.

Default: 0

**NLPsolver** *(string)*: NLP solver for sub-problems and . option file number

*solver[n]* Solver is the name of the GAMS NLP solver that should be used in the root node, and *n* is the integer corresponding to optfile. If .n is missing, the optfile is treated as zero, i.e., the NLP solver will not look for an options file. This option can be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: GAMS NLP solver

**reslim** *(real)*: Time limit for AlphaECP (in seconds)

Default: GAMS reslim

**solvelink** *(integer)*: Solvelink for NLP and MIP sub-solver

Default: 5

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Call NLP and MIP solver via script.</td>
</tr>
<tr>
<td>2</td>
<td>Call NLP and MIP solver via module.</td>
</tr>
<tr>
<td>5</td>
<td>Call NLP and MIP solver in memory.</td>
</tr>
</tbody>
</table>

**solvetrace** *(string)*: Filename of solvetrace file

**solvetracetime** *(real)*: Time interval when a trace record is written

Default: 1

**TOLepsf** *(real)*: Pseudo-convex objective function termination tolerance

Maximum allowed absolute difference between the nonlinear and the MIP objective function value (used only in ECPstrategy 3).

Range: [1e-20, 1]

Default: 1e-3

**TOLepsg** *(real)*: Constraint tolerance
The nonlinear constraint tolerance defines the maximum value that a nonlinear constraint may violate. For example, a constraint required to be zero may hold a value $\pm TOL_{e epsg}$ at a solution.

Range: $[1e-20, 1]$  
Default: $1e-3$

**TOL_{epsz}** (*real*): Distance tolerance for a new linearization

The maximum perpendicular distance between a valid cutting plane and its generation point (MIP solution).

Range: $[1e-20, 1]$  
Default: $1e-1$

**TOL_{grad}** (*real*): Gradient tolerance

The absolute value of a gradient's partial derivative must be above $TOL_{grad}$ value in order for it to be considered nonzero.

Range: $[1e-20, 1]$  
Default: $1e-6$

**TOL_{infbnd}** (*real*): Infinity bound (MIP variable bound)

All variables must have a positive and a negative finite bound in order to ensure a bounded MIP problem. The finite bound value $TOL_{infbnd}$ will be applied to single or double unbounded variables.

Default: $1e10$

**TOL_{optcr}** (*real*): Relative termination tolerance for MINLP

The relative objective gap termination criterion is satisfied if $|UB-LB|/ (10^{E-12}+\max(|LB|,|UB|)) < TOL_{optcr}$, where UB is the current best upper bound and LB is the current best lower bound. Upper bounds are obtained from solving NLP problems and lower bounds are obtained from lower bounds of MIP problems. If the inequality holds true and ECPtoltype=2, the algorithm terminates.

Range: $[1e-20, \infty]$  
Default: GAMS optCR

### 5.3.6 FAQ

- **What are good settings to solve a convex problem?**
  
  Use ECPmaster 1.

- **What are good settings if the solution speed is essential?**
  
  Try ECPstrategy 1 and CUTdeltcrit 1 to see if using multiple threads for the MIP solver improves the solution speed. However, there is considerable chance that a feasible solution for a non-convex problem with nonlinear equality constraints cannot be found.

- **What are good settings when the solution quality is essential?**
  
  Use NLPcalliter 1 and MIPsolstrat 4 or 3, and also try different values, for CUTnrcuts option, for example 0.1.

- **The objective function is non-linear, should the default ECPstrategy be used?**
  
  If the objective function constraint can be written in the required form of ECPstrategy 3 then this strategy may find a better solution. If the constraints and the objective function are pseudo-convex the global optimal solution will be found.
5.4 ANTIGONE

Author

Christodoulos A. Floudas, floudas@titan.princeton.edu; Computer-Aided Systems Laboratory; Department of Chemical and Biological Engineering; Princeton University

Ruth Misener, r.misener@imperial.ac.uk; Centre for Process Systems Engineering; Imperial College London

Date

16 April 2013

Version

ANTIGONE 1.0

5.4.1 Introduction

ANTIGONE, Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations, is a
deterministic general mixed-integer nonlinear global optimization framework [134] [135] [137] [136] [138].

MINLP is defined:

$$\begin{align*}
\text{min} & \quad f_0(x, y, z) \\
\text{s.t.} & \quad b_m^{\text{LO}} \leq f_m(x, y, z) \leq b_m^{\text{UP}} \quad \forall \ m \in \{1, \ldots, M\} \\
& \quad x \in \mathbb{R}^C; \ y \in \{0, 1\}^B; \ z \in \mathbb{Z}^I
\end{align*}$$

where $C$, $B$, $I$, and $M$ represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Parameters vectors $b_m^{\text{LO}}$ and $b_m^{\text{UP}}$ bound the constraints. We assume that it is possible to infer finite bounds $[x^L_i, x^U_i]$ on the variables participating in nonlinear terms $f_m$ and that the image of $f_m$ is finite on $x$. Typical expressions for $f_0(x, y, z)$ and $f_m(x, y, z)$ are:

$$f_m(x, y, z) = c_m + a_m^T [x; y; z] + [x; y; z]^T Q_m [x; y; z]$$

$$+ \sum_{s=1}^{S_m} c_{s,m} \cdot \prod_{c=1}^C x^{p_{s,m,c}}_c + \sum_{e=1}^{E_m} c_{e,m} \cdot e^x + \sum_{l=1}^{L_m} c_{l,m} \cdot \log x$$

where the powers $p_{s,m,c}$ are constant reals; $c_m$, $a_m$, $Q_m$, $c_{s,m}$, $c_{e,m}$, $c_{l,m}$ are constant coefficients; $S_m$, $E_m$, $L_m$ are the number of signomial, exponential, and logarithmic terms, respectively.

As illustrated in Figure 5.1, ANTIGONE responds dynamically to exploit special structure within (MINLP). ANTIGONE falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MINLP that rigorously bound the global solution; finds feasible solutions via local optimization; divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [68] [67].
5.4 ANTIGONE

Figure 5.1 Given an MINLP optimization problem, ANTIGONE reformulates the model, detects special structure in the reformulated MINLP, solves the optimization problem, and returns the model with respect to the original problem variables.

5.4.1.1 Licensing and software requirements

Using GAMS/ANTIGONE requires

1. an ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

5.4.1.2 Running GAMS/ANTIGONE

GAMS/ANTIGONE solves: NLP; MINLP; RMINLP; QCP; MIQCP; RMIQCP; CNS. If GAMS/ANTIGONE is not the default solver for these models, it can be called using the following command before the solve statement:

```plaintext
option nlp=antigone, minlp=antigone, rminlp=antigone;
```

5.4.2 GAMS/ANTIGONE Output

The log output shown below is generated using the MINLP model cecil_13 from the MINLPLib.

```
-------------------------------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.0
Ruth Misener and Christodoulos A. Floudas
Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University
-------------------------------------------------------------------

Before Pre-processing:
  840 Variables
    660 Continuous
    180 Binary
  929 Equations

After Pre-processing:
  520 Variables
```
418 Continuous
102 Binary
499 Equations
291 Linear
208 Nonconvex nonlinear
232 Nonlinear Terms
232 Signomial
730 Possible Reformulation Linearization Technique (RLT) equations
34 RLT Equations Added Outright to Formulation

Constituent Libraries:
- CPLEX Solving relaxations
- CONOPT Finding feasible points
- LAPACK Addressing linear systems
- Boost Bounding Intervals

Time (s) Nodes explored Nodes remaining Best possible Best found Relative Gap

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>1</td>
<td>1</td>
<td>-1.158e+05</td>
<td></td>
<td>+1.032e-03</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+4.337e-04</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.334e-04</td>
<td></td>
</tr>
<tr>
<td>258</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.140e-06</td>
<td></td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+4.091e-06</td>
<td></td>
</tr>
<tr>
<td>413</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+4.011e-06</td>
<td></td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>483</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+4.001e-06</td>
<td></td>
</tr>
<tr>
<td>571</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+3.959e-06</td>
<td></td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>640</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td></td>
<td>+3.880e-06</td>
<td></td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>702</td>
<td>1</td>
<td>0</td>
<td>-1.157e+05</td>
<td></td>
<td>+1.000e-06</td>
<td></td>
</tr>
</tbody>
</table>

Termination Status : Global minimum
Best Feasible Point: -1.156565e+05
Best Possible Point: -1.156566e+05
Relative Gap: +1.000000e-06
Algorithm analysis:

- 0 Nodes explored
- 0 Nodes remaining
- 0 Maximum tree depth

183 Cutting Planes (183 Globally Valid)
183 Signomial

702.38 Total time (CPU s)
- 0.07 Pre-processing
- 698.56 Solving MILP relaxations
- 0.95 Searching for feasible solutions
- 2.79 Variable bounds tightening
- 2.31 OBBT
- 1.48 FBBT (0.13 EC; 0.92 RLT; 0.00 Factoring)
- 0.00 Branching
5.4 ANTIGONE

5.4.3 Summary of ANTIGONE Options

5.4.3.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_opt_tol</td>
<td>absolute stopping tolerance</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>dumpsolutions</td>
<td>name of solutions index gdx file for writing alternate solutions</td>
<td></td>
</tr>
<tr>
<td>max_number_nodes</td>
<td>node limit</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td>max_time</td>
<td>resource limit</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>readparams</td>
<td>read secondary option file in ANTIGONE syntax</td>
<td></td>
</tr>
<tr>
<td>rel_opt_tol</td>
<td>relative stopping tolerance</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>trydual</td>
<td>call CONOPT or SNOPT to produce duals</td>
<td>5</td>
</tr>
</tbody>
</table>

5.4.3.2 Options for Solving the MILP Relaxations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplex_optfile</td>
<td>read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve</td>
<td></td>
</tr>
<tr>
<td>cut_generation_epsilon</td>
<td>absolute violation threshold for separating hyperplanes</td>
<td>1e-4</td>
</tr>
<tr>
<td>nominal_time_limit</td>
<td>nominal time limit for solving MILP subproblems</td>
<td>100</td>
</tr>
<tr>
<td>populate_solution_pool</td>
<td>emphasis on generating starting points</td>
<td>3</td>
</tr>
</tbody>
</table>

5.4.3.3 Options for Finding Feasible Solutions

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conopt_optfile</td>
<td>read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve</td>
<td></td>
</tr>
<tr>
<td>feas_soln_time_limit</td>
<td>time limit (s) for an NLP solve</td>
<td>30</td>
</tr>
<tr>
<td>feas_tolerance</td>
<td>absolute feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>nlp_solver</td>
<td>use CONOPT or SNOPT to find feasible solutions</td>
<td>conopt</td>
</tr>
</tbody>
</table>

5.4.3.4 Options for Branching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching_bounds_push_away</td>
<td>branch a minimum fraction away from the variable bounds</td>
<td>0.1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching_weight</td>
<td>branch on a convex combination of midpoint and solution</td>
<td>0.25</td>
</tr>
<tr>
<td>num_reliability_tests</td>
<td>number of strong branching initialization tests</td>
<td>8</td>
</tr>
<tr>
<td>reliability_branching</td>
<td>heuristic choice for building reliable pseudocosts</td>
<td>error</td>
</tr>
<tr>
<td>reliability_branching_mu</td>
<td>score parameter for building reliability</td>
<td>0.15</td>
</tr>
<tr>
<td>use_reliability_branching</td>
<td>use reliability branching?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.4.3.5 Options for Bounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbbt_improvement_bound</td>
<td>bounds reduction improvement threshold needed to exit FBBT loop</td>
<td>0.999</td>
</tr>
<tr>
<td>max_fbbt_iterations</td>
<td>maximum number of FBBT iterations</td>
<td>50</td>
</tr>
<tr>
<td>max_obbt_iterations</td>
<td>maximum number of OBBT iterations</td>
<td>30</td>
</tr>
<tr>
<td>max_time_each_obbt</td>
<td>time limit (s) for each OBBT LP</td>
<td>10</td>
</tr>
<tr>
<td>obbt_improvement_bound</td>
<td>bounds reduction improvement threshold</td>
<td>0.95</td>
</tr>
<tr>
<td>use_obbt</td>
<td>use optimality-based bounds tightening?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.4.3.6 Options for Logging to the Console

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_freq</td>
<td>how often should we log progress to the console?</td>
<td>5</td>
</tr>
<tr>
<td>logging_level</td>
<td>logging information level</td>
<td>-1</td>
</tr>
<tr>
<td>print_options</td>
<td>print the option parameter choices used in a single run?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.4.3.7 Options for Addressing Special Structure

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptive_add_rlt</td>
<td>use the dynamic approach to adaptively determine deep RLT cuts?</td>
<td>1</td>
</tr>
<tr>
<td>adaptive_add_rlt_tree_depth</td>
<td>tree depth for heuristic that adaptively determines deep RLT cuts</td>
<td>3</td>
</tr>
<tr>
<td>add_bilinear_terms</td>
<td>allow addition of nonconvex bilinear terms to generate deep RLT cuts</td>
<td>1</td>
</tr>
<tr>
<td>convexity_cuts</td>
<td>derive convexity-based separating cuts for multivariate terms?</td>
<td>1</td>
</tr>
<tr>
<td>dominant_ec_only</td>
<td>add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?</td>
<td>1</td>
</tr>
<tr>
<td>eigenvector_projections</td>
<td>use eigenvector projections as additional cuts?</td>
<td>1</td>
</tr>
<tr>
<td>eigenvector_projection_partitioning</td>
<td>allow partitioning on eigenvector projections?</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.4 ANTIGONE

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>low_dim_edge_concave_agg</code></td>
<td>use low-dimension edge-concave aggregations?</td>
<td>1</td>
</tr>
<tr>
<td><code>max_partitioned_quantities</code></td>
<td>number of partitioned quantities</td>
<td>0</td>
</tr>
<tr>
<td><code>max_rlt_cuts</code></td>
<td>maximum number of violated RLT cuts to add before resolving the relaxation?</td>
<td>100</td>
</tr>
<tr>
<td><code>naive_add_ec</code></td>
<td>naively integrate all low-dimension edge-concave aggregations into relaxations?</td>
<td>0</td>
</tr>
<tr>
<td><code>naive_add_rlt</code></td>
<td>naively add all RLT cuts to the relaxations?</td>
<td>0</td>
</tr>
<tr>
<td><code>number_of_partitions</code></td>
<td>how many partitions per variable?</td>
<td>1</td>
</tr>
<tr>
<td><code>partitioning_scheme</code></td>
<td>Partitioning scheme can be linear or logarithmic</td>
<td>linear</td>
</tr>
<tr>
<td><code>piecewise_linear_partitions</code></td>
<td>use piecewise-linear partitioning?</td>
<td>0</td>
</tr>
<tr>
<td><code>rlt</code></td>
<td>find RLT variable/equation and equation/equation pairs?</td>
<td>1</td>
</tr>
<tr>
<td><code>use_alpha_bb</code></td>
<td>apply globally-valid alphaBB cuts to tighten a node relaxation</td>
<td>1</td>
</tr>
<tr>
<td><code>use_edge_concave_dynamic</code></td>
<td>apply locally-valid edge-concave cuts to tighten a node relaxation</td>
<td>1</td>
</tr>
</tbody>
</table>

#### 5.4.4 Detailed Descriptions of ANTIGONE Options

**abs_opt_tol** (*real*): absolute stopping tolerance

Default: GAMS `optca`

**adaptive_add_rlt** (*boolean*): use the dynamic approach to adaptively determine deep RLT cuts?

In the first few levels of the branch-and-bound tree, query the RLT equations after solving an initial relaxation. Add violated equations to the relaxation and resolve. Track the most commonly-violated equations and include those cuts in later nodes.

Default: 1

**adaptive_add_rlt_tree_depth** (*integer*): tree depth for heuristic that adaptively determines deep RLT cuts

To the specified tree depth, solve the relaxation of a node twice if RLT equations are violated. After this depth, automatically add the most commonly violated cuts to the solution of each node.

Range: \(\{1, ..., 100\}\)

Default: 3

**add_bilinear_terms** (*boolean*): allow addition of nonconvex bilinear terms to generate deep RLT cuts

Default: 1

**branching_bounds_push-away** (*real*): branch a minimum fraction away from the variable bounds

Range: \([0, 0.5]\)

Default: 0.1
**branching_weight** *(real)*: branch on a convex combination of midpoint and solution

The branching weight specifies the emphasis on the midpoint of a variable, so larger branching weights imply branching closer to the center of a variable range.

Range: \([0, 1]\)

Default: 0.25

**conopt_optfile** *(string)*: read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve

Gain direct access to the GAMS/CONOPT options. The value of the string should match the name of the GAMS/CONOPT options file.

**convexity_cuts** *(boolean)*: derive convexity-based separating cuts for multivariable terms?

Default: 1

**cplex_optfile** *(string)*: read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve

Gain direct access to the GAMS/CPLEX options. Specifying an options file allows, for example, the possibility of running the CPLEX subsolver with multiple threads. The value of the string should match the name of the GAMS/CPLEX options file.

**cut_generation_epsilon** *(real)*: absolute violation threshold for separating hyperplanes

Absolute violation threshold to generate separating hyperplanes for convex multivariable terms

Range: \([1e^{-7}, 10]\)

Default: 1e-4

**dominant_ec_only** *(boolean)*: add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?

Default: 1

**dumpsolutions** *(string)*: name of solutions index gdx file for writing alternate solutions

The GDX file specified by this option will contain a set call `index` that contains the names of GDX files with the individual solutions. For details see example model `dumpsol` in the GAMS Test Library.

**eigenvector_projections** *(boolean)*: use eigenvector projections as additional cuts?

Default: 1

**eigenvector_projection_partitioning** *(boolean)*: allow partitioning on eigenvector projections?

Default: 1

**fbbt_improvement_bound** *(real)*: bounds reduction improvement threshold needed to exit FBBT loop

Range: \([0, 1]\)

Default: 0.999
feas_soln_time_limit (real): time limit (s) for an NLP solve

    Range: [1, ∞]
    Default: 30

feas_tolerance (real): absolute feasibility tolerance

    Default: 1e-6

logging_freq (real): how often should we log progress to the console?

    Wait at least the specified time in seconds before next output to the console
    Range: [1, ∞]
    Default: 5

logging_level (integer): logging information level

    Log to the console at the specified level (-1: default; 0: minimal logging; 3: extensive logging)
    Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>minimal plus warnings</td>
</tr>
<tr>
<td>0</td>
<td>minimal</td>
</tr>
<tr>
<td>1</td>
<td>entering info</td>
</tr>
<tr>
<td>2</td>
<td>updating info</td>
</tr>
<tr>
<td>3</td>
<td>includes Cplex updates</td>
</tr>
</tbody>
</table>

low_dim_edge_concave_agg (boolean): use low-dimension edge-concave aggregations?

    Default: 1

max_fbbt_iterations (integer): maximum number of FBBT iterations

    Range: {1, ..., 100}
    Default: 50

max_number_nodes (integer): node limit

    Default: GAMS nodlim

max_obbt_iterations (integer): maximum number of OBBT iterations

    Range: {1, ..., 100}
    Default: 30

max_partitioned_quantities (integer): number of partitioned quantities

    Range: {0, ..., 50}
    Default: 0
**max_rlt_cuts (integer):** maximum number of violated RLT cuts to add before resolving the relaxation?

Range: \(\{1, \ldots, 1000\}\)

Default: 100

**max_time (real):** resource limit

Default: GAMS reslim

**max_time_each_obbt (real):** time limit (s) for each OBBT LP

Range: \([1, 100]\)

Default: 10

**naive_add_ec (boolean):** naively integrate all low-dimension edge-concave aggregations into relaxations?

Default: 0

**naive_add_rlt (boolean):** naively add all RLT cuts to the relaxations?

Default: 0

**nlp_solver (string):** use CONOPT or SNOPT to find feasible solutions

Note, that independent of the setting for this option, for the initial NLP solve from the user provided starting point, always CONOPT is used, if available. Further, for the final NLP solve (see trydual), always CONOPT is used, if available, otherwise SNOPT is used.

Default: conopt

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>conopt</td>
<td>Conopt</td>
</tr>
<tr>
<td>snopt</td>
<td>Snopt</td>
</tr>
</tbody>
</table>

**nominal_time_limit (real):** nominal time limit for solving MILP subproblems

Nominal time limit for solving MILP subproblems. Terminate long-running MILP subproblems over this time limit once they reach an integer feasible point

Range: \([0.1, 1000]\)

Default: 100

**number_of_partitions (integer):** how many partitions per variable?

Range: \(\{0, \ldots, 16\}\)

Default: 1

**num_reliability_tests (integer):** number of strong branching initialization tests
5.4 ANTIGONE

Range: \{1, ..., 100\}

Default: 8

**obbt_improvement_bound** *(real)*: bounds reduction improvement threshold

Bounds reduction improvement threshold needed to exit OBBT loop. This parameter also determines whether to continue obbt in child; if the parent bound improvement is less than this threshold, then child node won’t try OBBT.

Range: \([0, 1]\)

Default: 0.95

**partitioning_scheme** *(string)*: Partitioning scheme can be linear or logarithmic

Linear partitioning uses a number of binary variables linear in the number of partitions while logarithmic partitioning uses a number of binary variables logarithmic in the number of breakpoints. Linear partitioning tends to be numerically favorable for a few breakpoints while logarithmic partitioning is better for a larger number of breakpoints.

Default: linear

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>Linear partitioning</td>
</tr>
<tr>
<td>logarithmic</td>
<td>Logarithmic partitioning</td>
</tr>
</tbody>
</table>

**piecewise_linear_partitions** *(boolean)*: use piecewise-linear partitioning?

Default: 0

**populate_solution_pool** *(integer)*: emphasis on generating starting points

Emphasis on generating many starting points for NLP solves using the CPLEX solution pool feature. Larger number implies more starting points.

Range: \{0, ..., 4\}

Default: 3

**print_options** *(boolean)*: print the option parameter choices used in a single run?

Default: 1

**readparams** *(string)*: read secondary option file in ANTIGONE syntax

**reliability_branching** *(string)*: heuristic choice for building reliable pseudocosts

Default: error

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>Max Error Branching</td>
</tr>
<tr>
<td>forward</td>
<td>Forward branching</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverse branching</td>
</tr>
</tbody>
</table>
reliability_branching_mu (real): score parameter for building reliability←

Range: [0, 1]
Default: 0.15

rel_opt_tol (real): relative stopping tolerance←

Default: GAMS optcr

rlt (boolean): find RLT variable/equation and equation/equation pairs?←

Default: 1

trydual (real): call CONOPT or SNOPT to produce duals←

Spend the specified amount of time in seconds or less in producing a dual solution by calling CONOPT or SNOPT.
Range: [0, ∞]
Default: 5

use_alpha_bb (boolean): apply globally-valid alphaBB cuts to tighten a node relaxation←

Default: 1

use_edge_concave_dynamic (boolean): apply locally-valid edge-concave cuts to tighten a node relaxation←

Default: 1

use_obbt (boolean): use optimality-based bounds tightening?←

Default: 1

use_reliability_branching (boolean): use reliability branching?←

Default: 1

5.4.5 ANTIGONE Algorithmic Features

As illustrated in Figure 5.1, the primary algorithmic features in ANTIGONE are reformulating model input, elucidating special structure, and branch-and-bound global optimization [134] [135] [137] [136] [138] .

5.4.5.1 Reformulating Model Input

As illustrated in Figure 5.2, ANTIGONE transforms a factorable programming tree into a flattened expression tree to capitalize on the development of tight convex underestimators for specific classes of nonlinear terms. ANTIGONE extends the efficacy of hybrid strategies by meaningfully integrating mutually reinforcing operator- and term-based strategies [27] [76] [136] . This approach reformulates towards multivariable terms with specialized underestimators while maintaining a tree-like representation of powers that cannot be distributed and convex operators that can be exploited by dynamic cut generation.
5.4 ANTIGONE

5.4.5.2 Elucidating Special Structure

After reformulating the user-defined MINLP, ANTIGONE detects special mathematical structure that it will exploit in the branch-and-cut phase (Section Branch-and-Bound Global Optimization). The types of special structure that ANTIGONE considers are: reformulation-linearization technique (RLT) equations; convexity/concavity; edge-convexity/edge-concavity; \( \alpha \)BB relaxations; term-specific underestimators [134] [135] [136] [137] [138] .

- RLT multiplies every pairwise combination of: variables; nonlinear terms; equations [11] [119] [135] [136] [138] [169] [170] [171] [172] [173] . ANTIGONE saves the combinations that do not introduce new terms into the model formulation and updates these equations at every node of the branch-and-cut tree. Special RLT equations are added directly to the model formulation; other RLT equations are used as cutting planes and integrated into the feasibility-based bounds tightening routines.

- Convexity/Concavity permits the easy generation of a cutting plane at a point \( \hat{x} \):

\[
\begin{align*}
  f(x) &\geq f(\hat{x}) + f'(\hat{x}) \cdot (x - \hat{x}) \quad \text{(convex)} \\
  f(x) &\leq f(\hat{x}) + f'(\hat{x}) \cdot (x - \hat{x}) \quad \text{(concave)}
\end{align*}
\]

Based on interval arithmetic, terms and multi-term expressions are labeled as always/sometimes/never convex/ concave; this information is used in the branch-and-cut phase.

- Edge-Convexity/Edge-Concavity implies a vertex polyhedral envelope; ANTIGONE labels terms and multi-term expressions as always/sometimes/never edge-convex/edge-concave with a simple interval arithmetic test [133] [175] [176] [177] .

- \( \alpha \)BB underestimators convexify an expression with univariate quadratics [5] [6] [10] [66] [128]; ANTIGONE uses \( \alpha \)BB to relax aggregates of bilinear terms.

- Term-Specific Underestimators are diagrammed in Figure 5.3; their implementation is based on work available in the open literature [53] [52] [65] [77] [118] [125] [123] [124] [129] [133] [134] [146] [175] [176] [178] .
5.4.5.3 Branch-and-Bound Global Optimization

After the reformulation and special structure detection phases, ANTIGONE initiates a branch-and-cut global optimization algorithm that generates tight convex underestimators, dynamically generates separating hyperplanes, bounds the variables \([6] [10] [5] [11] [18] [49] [50] [116] [157] [158] [172] [173] [191]\); branches on the search space \([3] [18]\), and finds feasible solutions.

5.5 BARON

Author

Nick Sahinidis, The Optimization Firm, LLC, niksah@minlp.com, http://www.minlp.com

Date

5 June 2015

5.5.1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a GAMS solver for the global solution of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP).

While traditional NLP and MINLP algorithms are guaranteed to converge only under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are guaranteed to provide global optima under fairly general assumptions. These include the existence of finite lower and upper bounds on nonlinear expressions in the NLP or MINLP to be solved.

BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation, interval analysis, and duality techniques for reducing ranges of variables in the course of the algorithm. Rigorous relaxations are constructed by enlarging the feasible region and/or underestimating the objective function.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign. The algorithms implemented in the software, the theory behind them, and some related applications are (partly) described in \([157] [51] [158] [161] [93] [120] [78] [187] [168] [188] [163] [79] [159] [178] [179] [180] [183] [184] [165] [160] [162] [7] [181] [164] [182] [36] [12] [149] [13] [106] [107] [208] [209] [108] [210] [14] \).
5.5 BARON

5.5.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license. BARON comes with several embedded LP/MIP and NLP solvers (CBC; IPOPT, FilterSD, FilterSQP). Additionally, GAMS/BARON users can expedite convergence by accessing CPLEX and XPRESS to solve BARON's LP/MIP subproblems and MINOS, SNOPT, and any GAMS NLP solver, such as CONOPT, to solve BARON's NLP subproblems. These solvers need to be licensed with GAMS separately.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and select an NLP solver automatically. The user can use the options LPSol and NLPSol to specify the LP or NLP solver. If the user does not have a license for the user-specified LP and NLP solver, BARON will automatically select a licensed LP and NLP solver and may default to CLP and IPOPT if no other LP and NLP solver is available, respectively. BARON can be used without a local NLP solver by setting DoLocal and NumLoc to 0.

5.5.1.2 Running GAMS/BARON

BARON is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP. If BARON is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```plaintext
option <modeltype>=baron;
```

where `<modeltype>` stands for LP, MIP, RMIP, QCP, MIQCP, RMIQCP, CNS, NLP, DNLP, MINLP, or RMINLP.

5.5.2 Model requirements

In order to achieve convergence to global optimality, additional model constraints may be required. The additional constraints may speed up solver time and increase the probability of success.

5.5.2.1 Variable and expression bounds

All nonlinear expressions in the mathematical program to be solved must be bounded below and/or above. It is important that finite lower and upper bounds be provided by the user on all problem variables. Note that providing finite bounds on variables alone may not suffice to guarantee finite bounds on nonlinear expressions arising in the model.

For example, consider the term 1/x for x ∈ [0,1], which has finite variable bounds, but is unbounded. It is important to provide bounds for problem variables that guarantee that the problem functions are finitely-valued. If the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued, BARON will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided is not guaranteed. Occasionally, because of the lack of bounds no numerically stable lower bounding problems can be constructed, in which case BARON may terminate.

See section Some BARON features on how to specify variable bounds.

5.5.2.2 Allowable nonlinear functions

In addition to multiplication and division, GAMS/BARON can handle nonlinear functions that involve \( \exp(x) \), \( \ln(x) \), \( x^\alpha \) for real \( \alpha \), \( x^\beta \) for real \( \beta \), \( x^2 \), and \( |x| \). Currently, there is no support for other functions, including the trigonometric functions \( \sin(x) \), \( \cos(x) \), etc.
5.5.3 BARON output

5.5.3.1 BARON log output

The log output below is obtained for the MINLP model gear.gms from the GAMS model library using absolute and relative optimality tolerances of 1e-6.

===========================================================================
BARON version 15.6.5. Built: LNX-64 Fri Jun 5 08:34:09 EDT 2015

If you use this software, please cite:

BARON is a product of The Optimization Firm, LLC. http://www.minlp.com/
Parts of the BARON software were created at the University of Illinois at Urbana-Champaign.

===========================================================================
This BARON run may utilize the following subsolver(s)
For LP/MIP: ILOG CPLEX
For NLP: MINOS, SNOPT, GAMS external NLP, COIN IPOPT with MUMPS and METIS, FILTERSD

===========================================================================
Starting solution is feasible with a value of 36.1767610000
Doing local search
Solving bounding LP
Starting multi-start local search
Preprocessing found feasible solution with value 4.23791612465
Done with local search

===========================================================================
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Open nodes</th>
<th>Time (s)</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.02</td>
<td>1.00000</td>
<td>4.23792</td>
</tr>
<tr>
<td>*</td>
<td>1</td>
<td>0.02</td>
<td>1.00000</td>
<td>3.29321</td>
</tr>
<tr>
<td>*</td>
<td>1</td>
<td>0.02</td>
<td>1.00000</td>
<td>2.20487</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.02</td>
<td>1.00000</td>
<td>2.00487</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
<td>0.03</td>
<td>1.00000</td>
<td>1.01273</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
<td>0.03</td>
<td>1.00000</td>
<td>1.00117</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
<td>0.03</td>
<td>1.00000</td>
<td>1.00018</td>
</tr>
<tr>
<td>*</td>
<td>14</td>
<td>0.04</td>
<td>1.00000</td>
<td>1.00004</td>
</tr>
<tr>
<td>*</td>
<td>30</td>
<td>0.05</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>0.05</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

Cleaning up

*** Normal completion ***

Wall clock time: 0.05
Total CPU time used: 0.05

Total no. of BaR iterations: 30
Best solution found at node: 30
Max. no. of nodes in memory: 11

All done

===========================================================================
The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 36.1767610000. BARON subsequently performs a randomized local search procedure and, eventually, finds a feasible solution with an objective of 4.23791612465. Then, the iteration log provides information every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed at the end of the root node, whenever the value of the incumbent is improved by at least $10^{-5}$, and at the end of the search. A star (*) in the first position of a line indicates that a better feasible solution was found that improves the value of previous best known solution by at least $10^{-5}$. The log fields include the iteration number, number of open branch-and-bound nodes, the CPU time taken thus far, the lower bound, and the upper bound for the problem. The log output fields are summarized below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itn. no.</td>
<td>The number of the current iteration. A plus (+) following the iteration number denotes reporting while solving a probing (as opposed to a relaxation) subproblem of the corresponding node.</td>
</tr>
<tr>
<td>Open Nodes</td>
<td>Number of open nodes in branch-and-reduce tree.</td>
</tr>
<tr>
<td>Time</td>
<td>Current computational time in seconds. CPU time is reported for single-threaded jobs and wall clock time is reported for multi-threaded jobs.</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>Current lower bound on the model.</td>
</tr>
<tr>
<td>Upper Bound</td>
<td>Current upper bound on the model.</td>
</tr>
</tbody>
</table>

Once the branch-and-reduce tree is searched, the best solution is isolated and a corresponding dual solution is calculated. Finally, the total number of branch-and-reduce iterations (number of search tree nodes) is reported, followed by the node where the best solution was identified (a -1 indicates preprocessing as explained in the next section on termination messages).

### 5.5.3.2 Termination messages, model and solver statuses

Upon normal termination, BARON will report the node where the optimal solution was found. We refer to this node as `nodeopt`. Associated with this node is a return code indicating the status of the solution found at `nodeopt`. The return code is given in the log line:

**Best solution found at node: (return code)**

The return codes have the following interpretation:

- `-3` : no feasible solution found
- `-2` : the best solution found was the user-supplied
- `-1` : the best solution was found during preprocessing
- `i` : the best solution was found in the i-th node of the tree

In addition to reporting `nodeopt`, upon termination, BARON will issue one of the following statements:

*** Normal completion ***

This is the most desirable termination status. The problem has been solved within tolerances in this case. If BARON returns a code of -3, then no feasible solution exists.

*** Heuristic termination ***
While global optimality is not guaranteed in this case, BARON will terminate with this message when (a) a feasible solution has been found and (b) the progress of lower/upper bounds satisfies the heuristic termination criterion set by the user through BARON's *DeltaTerm* option.

*** User did not provide appropriate variable bounds ***

The user will need to read the BARON output (in file *sum.dat* in the scratch directory, use GAMS parameter *keep=1* to prevent the automatic removal of this directory) for pointers to variables and expressions with missing bounds. The model should be modified in order to provide bounds for variables and intermediate expressions that make it possible for BARON to construct reliable relaxations. Even though relaxation bounds are printed on the screen to give the user a feeling for convergence, these bounds may not be valid for the problem at hand. This message is followed by one of the following two messages:

*** Infeasibility is therefore not guaranteed ***

This indicates that, because of missing bounds, no feasible solution was found, but model infeasibility was not proven.

*** Globality is therefore not guaranteed ***

This indicates that, because of missing bounds, a feasible solution was found, but global optimality was not proven.

*** Max. allowable nodes in memory reached ***

The user will need to increase the physical memory of the computer or change algorihmic options, such as branching and node selection rules, to reduce the size of the search tree and memory required for storage.

*** Max. allowable BarR iterations reached ***

The user will need to increase the maximum number of allowable iterations. The BARON option is *MaxIter*. To specify this in GAMS, one can use the *NodLim* option. We remark that the BARON option *MaxIter* overrides *NodLim*.

*** Max. allowable CPU time exceeded ***

The user will need to increase the maximum of allowable CPU time. The BARON option is *MaxTime*. To specify this in GAMS, one can use the *ResLim* option. We remark that the BARON option *MaxTime* overrides *ResLim*.

*** Problem is numerically sensitive ***

BARON is designed to automatically handle problems with numerical difficulties. However, for certain problems, the global optimum is numerically sensitive. This occurs, for instance, when the objective function value varies significantly over small neighborhoods of points that are strictly outside the feasible region but nonetheless feasible within numerical tolerances. When BARON returns this message, the "Best possible" reported on the objective is likely correct.

*** Search interrupted by user ***

The run was interrupted by the user (Ctrl-C).

*** Insufficient memory for data structures ***

More memory is needed to set up the problem data structures. The user will need to increase the physical memory available on the computer in order to accommodate problems of this size.

*** A potentially catastrophic access violation just took place ***

In the unlikely event of a memory access violation, BARON will terminate the search and return the best known solution. Please report problems that lead to this condition to Nick Sahinidis ([niksah@minlp.com](mailto:niksah@minlp.com)).
5.5.4 Some BARON features

The features described in this section rely on options that are further detailed in the next section. For details of the algorithmic implementations, the user may wish to consult publications cited at the end of this document.

5.5.4.1 No starting point is required

In contrast to many NLP algorithms that require a feasible starting point, a starting point is not required for BARON. A user may optionally provide a starting point for all or even some of the problem variables. BARON will judiciously initialize any variables that are not initialized by the user. Even when the problem functions cannot be evaluated at a user-provided starting point, BARON is still capable of carrying out its global search.

For problems for which GAMS compilation is aborted because the nonlinear functions cannot be evaluated at the starting point, the user can use the following commands before the SOLVE statement:

\[
\begin{align*}
\text{MaxExecError} & = 100000; \\
\text{option sys12 = 1; }
\end{align*}
\]

The first command asks GAMS to continue compilation for as many as MaxExecError execution errors. The sys12 option will pass the model to the BARON despite the execution errors. Even though the starting point is bad in this case, BARON is capable of carrying out its global search.

5.5.4.2 Finding a few of the best or all feasible solutions

BARON offers a facility, through its NumSol option, to find the best few, or even all feasible, solutions to a model. The development of this facility was motivated by combinatorial optimization problems but the facility is applicable to continuous problems as well. Even for combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. Instead, it utilizes a single search tree, thus providing a computationally efficient method for finding multiple solutions. Furthermore, because BARON's approach applies to integer as well as continuous programs, it can be used to find all feasible solutions to a system of nonlinear equality and inequality constraints.

Once a model is solved by BARON with the NumSol option, the solutions found can be recovered using the GAMS GDX facility. An example is provided below.

```
$eolcom !
$Ontext

Purpose: demonstrate use of BARON option 'numsol' to obtain the best numsol solutions of an optimization problem in a single branch-and-bound search tree.

The model solved here is a linear general integer problem with 18 feasible solutions. BARON is run with a request to find up to 20 solutions. The model solved is the same as the one solved in gamslib/icut.gms.

$Offtext

set i index of integer variables / 1 * 4 /

variables x(i) variables

z objective variable
```
integer variable x;

x.lo(i) = 2; x.up(i) = 4; x.fx('2') = 3;  ! fix one variable
x.up('4') = 3;  ! only two values

equation obj obj definition;
* pick an objective function which will order the solutions
obj .. z =e= sum(i, power(10,card(i)-ord(i))*x(i));

model enum / all /;
* instruct BARON to return numsol solutions
$onecho > baron.opt
numsol 20
gdxout multsol
$offecho
enum.optfile=1; option mip=baron, limrow=0, limcol=0, optca=1e-5,
optcr=1e-5; solve enum minimizing z using mip;

* recover BARON solutions through GDX
set sol /multsol1*multsol100/; variables xsol(sol,i), zsol(sol);
execute 'gdxmerge multsol*.gdx > %gams.scrdir%merge.%gams.scrext%';
execute_load 'merged.gdx', xsol=x, zsol=z;

option decimals=8;
display xsol.l, zsol.l;

5.5.4.3 Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to existence of multiple
local solutions, modelers often make use of multiple local searches from randomly generated starting points.
This can be easily done with BARON's NumLoc option, which determines the number of local searches to
be done by BARON's preprocessor. BARON can be forced to terminate after preprocessing by setting
the number of iterations to 0 through the MaxIter option. In addition to local search, BARON's preprocessor
performs extensive reduction of variable ranges. To sample the search space for local minima without range
reduction, one would have to set to 0 the range reduction options TDo, MDo, LBTTD0, and OBTTDo.
On the other hand, leaving these options to their default values increases the likelihood of finding high
quality local optima during preprocessing. If NumLoc is set to -1, local searches in preprocessing will be
done from randomly generated starting points until global optimality is proved or MaxTime seconds have elapsed.

5.5.4.4 Systematic treatment of unbounded problems

If BARON declares a problem as unbounded, it will search for and may report a vertex and direction of
an unbounded ray. In addition, BARON will report the best solution found. This will be a feasible point
that is as far as possible along an unbounded ray while avoiding numerical errors due to floating point
arithmetic.
5.5.4.5 Systematic treatment of infeasible problems

If BARON declares a problem as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. This, so-called, irreducibly inconsistent system (IIS) can be obtained by BARON for all types of problems handled by BARON, including linear and nonlinear, continuous and integer, convex and nonconvex, and problems with complementarity constraints. BARON's CompIIS option can be used to identify an IIS.

As an example, consider the problem of minimizing the nonconvex function \( x_1 x_3 \) over the following nonconvex constrained set:

\[
\begin{align*}
  e_1 : & \quad 85 + 0.006x_2 x_5 + 0.0006x_1 x_4 - 0.002x_3 x_5 \leq 92 \\
  e_2 : & \quad 0.8x_2 x_5 + 0.003x_1 x_2 + 0.002x_3^2 = 110 \\
  e_3 : & \quad 9 + 0.005x_3 x_5 + 0.001x_1 x_3 + 0.002x_4 x_1 \leq 25 \\
  & \quad 78 \leq x_1 \leq 102 \\
  & \quad 33 \leq x_2 \leq 45 \\
  & \quad 27 \leq x_i \leq 45, \quad i = 3, \ldots, 5
\end{align*}
\]

When this problem is solved with CompIIS equal to 1, BARON provides the following result in the listing file:

Number of equations in the IIS: 1.
Upper: \( e_2 \leq 110 \)

Number of variables in the IIS: 3.
Lower: \( x_1 \geq 78 \)
Lower: \( x_2 \geq 33 \)
Lower: \( x_5 \geq 27 \)

The IIS consists of the lower bounds of variables \( x_1, x_2, \) and \( x_5 \), along with the \( \leq \) part of the equality constraint \( e_2 \). This suggests that constraint \( e_2 \) and the entire model can be made feasible by lowering the lower bound of any of the three variables that are part of the IIS, whereas modifying the bounds of \( x_3 \) would not make the model feasible.

If a problem is known to be infeasible and the user desires to identify an IIS, it may be beneficial to set BARON’s NumLoc option to zero. Doing so will deactivate BARON's initial upper bounding search, which involves multiple local searches. On the other hand, DoLocal should be nonzero in order to permit local search during the solution of certain subproblems that BARON solves while searching for an IIS.

5.5.4.6 Handling of complementarity constraints

Complementarity relationships of the type \( f(x) g(x) = 0 \) are automatically recognized and exploited algorithmically by BARON. The functions \( f \) and \( g \) may be univariate or multivariate, linear or nonlinear, convex or nonconvex, in terms of continuous and/or integer variables, and may be subject to additional constraints in the model. These complementarity relationships can be inferred by BARON even when implied by problem constraints and variable bounds. As a result, BARON can solve general mathematical programs with equilibrium constraints (MPECs). This class of problems includes the classical linear complementarity problem

\[(LCP): \text{Find } z \geq 0 \text{ and } q \text{ such that }\]
\[ Mz + q \geq 0, \]
\[ z^t(Mz + q) = 0 \]

as well as the more general mixed complementarity problem

(MCP): Given a function \( f : \mathbb{R}^n \rightarrow \mathbb{R}^n \) and bounds \( l, u \in \mathbb{R}^n \) with \( \mathbb{R} = \mathbb{R} \cup \{-\infty, +\infty\} \), find \( z \in \mathbb{R}^n \) and \( w, v \in \mathbb{R}_+^n \) such that

\[ f(z) = w - v, \]
\[ l \leq z \leq u, \]
\[ (z - l)^t w = 0, \]
\[ (u - z)^t v = 0. \]

Both problems are automatically recognized and exploited by BARON without the user having to mark complementarities in any special way. In GAMS, all these problems can be solved by BARON when declared as NLP or MINLP models.

5.5.4.7 Parallel capabilities

For difficult problems with integer variables, most of BARON's time is spent on solving MIP relaxations. Hence, considerable speedups may be obtained via parallel solution of the MIP subproblems. For this purpose, the option threads may be used to specify the number of cores that BARON's MIP subsolver is allowed to use. By default, this option has the value of 1, meaning that a single core will be utilized.

5.5.5 The BARON options

The BARON options allow the user to control termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed in this section.

Many options can also be set in the GAMS model. The most relevant GAMS options are ResLim, NodLim, OptCA, OptCR, OptFile, and CutOff. The IterLim option is not implemented. Specify BARON iterations, the user can set the MaxIter option, which is equivalent to the GAMS option NodLim.

Additionally, a BARON Options file can be provided. See section The Solver Options File for general use of solver option files.

For branching, users can specify separate branching priorities for any discrete and continuous variables using Dot Options. To specify variable branching priorities, one specifies

\[(variable).prior(value)\]

in the BARON options file, where \((value)\) can be any non-negative real value. The lower the value, the higher the priority for branching, see also Setting Priorities for Branching. Specifying maxdouble for \((value)\) translates to passing 0 as branching priority for \((variable)\).

5.5.5.1 Termination options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsConFeasTol</td>
<td>Absolute constraint feasibility tolerance. This tolerance is used for general constraints and variable bounds. ( \text{AbsConFeasTol} ) must be ( \geq 1e^{-12} ). A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: ([1e^{-12}, \infty])</td>
<td>1e-5</td>
</tr>
<tr>
<td>AbsIntFeasTol</td>
<td>Absolute integer feasibility tolerance. All integer variable values must satisfy this tolerance. ( \text{AbsIntFeasTol} ) must be ( \geq 1e^{-12} ). A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: ([1e^{-12}, \infty])</td>
<td>1e-5</td>
</tr>
<tr>
<td>BoxTol</td>
<td>Box elimination tolerance. Boxes will be eliminated if smaller than this tolerance. ( \text{BoxTol} ) must be ( \geq 1e^{-12} ).</td>
<td>1e-8</td>
</tr>
<tr>
<td>CutOff</td>
<td>Eliminate solutions that are no better than this value. Can also be used with the GAMS model attribute option ( \text{CutOff} ).</td>
<td>GAMS CutOff</td>
</tr>
<tr>
<td>DeltaA</td>
<td>Absolute improvement for insufficient progress termination. ( \text{DeltaA} (\delta_a) ) must be ( \geq 1e^{-12} ). Range: ([1e^{-12}, \infty])</td>
<td>(\infty)</td>
</tr>
<tr>
<td>DeltaR</td>
<td>Relative improvement for insufficient progress termination. ( \text{DeltaR} (\delta_r) ) must be ( \geq 1e^{-12} ). Range: ([1e^{-12}, \infty])</td>
<td>1</td>
</tr>
<tr>
<td>DeltaT</td>
<td>Time interval for insufficient progress termination. If ( \text{DeltaTerm} ) is set to 1, BARON will terminate if insufficient progress is made over ( \text{DeltaT} (\delta_t) ) consecutive seconds. If ( \delta_t ) is set to a non-positive quantity, BARON will automatically set ( \delta_t ) equal to (-\delta_t ) times the CPU time taken till the end of root node processing. ( \text{DeltaT} ) can take any real value. Range: ([-\infty, \infty])</td>
<td>-100</td>
</tr>
<tr>
<td>DeltaTerm</td>
<td>Indicates whether insufficient progress termination is on or off. Users have the option to request BARON to terminate if insufficient progress is made over ( \text{DeltaT} (\delta_t) ) consecutive seconds. Progress is measured using the absolute and relative improvement thresholds ( \text{DeltaA} (\delta_a) ) and ( \text{DeltaR} (\delta_r) ). Termination will occur if, over a period of ( \delta_t ) consecutive seconds, the value of the best solution found by BARON is not improved by at least an absolute amount ( \delta_a ) or an amount equal to ( \delta_r ) times the value of the incumbent at time ( t-\delta_t ). This termination condition is enforced after processing the root node and only after a feasible solution has been obtained. Because it relies on CPU time measurements, which may depend on machine load, this option may result in nondeterministic behavior. 0: do not enforce this termination condition 1: terminate if progress is insufficient.</td>
<td>0</td>
</tr>
<tr>
<td>EpsA</td>
<td>Absolute termination tolerance. BARON terminates if (</td>
<td>U-L</td>
</tr>
<tr>
<td>EpsR</td>
<td>Relative termination tolerance. BARON terminates if (</td>
<td>U-L</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>FirstFeas</td>
<td>Changes the search for first <code>numsol</code> solutions. If set to 1, BARON will terminate once it finds <code>numSol</code> feasible solutions, irrespective of solution quality. 0: search for the best <code>numSol</code> feasible solutions 1: find <code>numSol</code> solutions irrespective of solution quality.</td>
<td>0</td>
</tr>
<tr>
<td>FirstLoc</td>
<td>Terminate the search as soon as a local optimum is found</td>
<td>0</td>
</tr>
<tr>
<td>ISolTol</td>
<td>Solution Distance. Separation distance between solutions. This option is used in conjunction with <code>numSol</code>. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an $l_\infty$ distance that does not exceed <code>IsolTol</code> will be treated as identical by BARON. <code>IsolTol</code> must be $\geq 1e-12$.</td>
<td>$1e-4$</td>
</tr>
<tr>
<td>MaxIter</td>
<td>Maximum number of branch-and-reduce iterations allowed. Setting <code>MaxIter</code> to 0 will force BARON to terminate after root node preprocessing. Setting <code>MaxIter</code> to 1 will result in termination after the solution of the root node. <code>MaxIter</code> must be $\geq -1$, where -1 implies unlimited. Range: ${-1, \ldots, \infty}$</td>
<td>GAMS NodeLim</td>
</tr>
<tr>
<td>MaxTime</td>
<td>Maximum time allowed (sec). <code>MaxTime</code> must be $\geq 0$, where 0 implies unlimited. For single-threaded jobs, i.e., when <code>threads</code> equals 1, this limit is enforced on CPU time consumed by the job. For multi-threaded jobs, the limit is enforced on wall clock time.</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>NumSol</td>
<td>Number of feasible solutions to be found. Solutions found will be listed in the <code>results</code> file. As long as <code>NumSol</code> $\neq -1$, these solutions will be sorted from best to worse. If <code>NumSol</code> is set to -1, BARON will search for all feasible solutions to the given model and print them, in the order in which they are found, in the <code>results</code> file. <code>NumSol</code> must be $\geq -1$. Range: ${-1, \ldots, \infty}$</td>
<td>1</td>
</tr>
<tr>
<td>RelConFeasTol</td>
<td>Relative constraint feasibility tolerance. This tolerance is used for general constraints and variable bounds. <code>RelConFeasTol</code> must be $\geq 0$. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: $[0, 0.1]$</td>
<td>0</td>
</tr>
<tr>
<td>RelIntFeasTol</td>
<td>Relative integer feasibility tolerance. All integer variable values must satisfy this tolerance. <code>RelIntFeasTol</code> must be $\geq 0$. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: $[0, 0.1]$</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.5.5.2 Relaxation options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nouter1</td>
<td>Number of outer approximators of convex univariant functions <code>Nouter1</code> must be $\geq 0$.</td>
<td>4</td>
</tr>
<tr>
<td>Nouter1</td>
<td>Number of rounds of cutting plane generation at node relaxation <code>Nouter1</code> must be a $\geq 0$.</td>
<td>4</td>
</tr>
<tr>
<td>NoutPerVar</td>
<td>Number of outer approximations per variable for convex multivariate functions <code>NoutPerVar</code> must be $\geq 0$.</td>
<td>4</td>
</tr>
</tbody>
</table>
### 5.5.5 BARON

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutGrid</td>
<td>Number of grid points per variable for convex multivariate approximators of BARON’s CONVEX.EQUATIONS. OutGrid must be a ≥ 0.</td>
<td>20</td>
</tr>
<tr>
<td>Threads</td>
<td>Number of cores used for solution of MIP subproblems. The value of this option is passed to CBC, CPLEX, and XPRESS. Range: {1, ..., ∞}</td>
<td>max(1,GAMS Threads)</td>
</tr>
</tbody>
</table>

#### 5.5.5.3 Range reduction options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBTTDo</td>
<td>Linear-feasibility-based range reduction option (poor man’s LPs) 0: no range reduction based on feasibility. 1: range reduction done based on feasibility.</td>
<td>1</td>
</tr>
<tr>
<td>MDo</td>
<td>Marginals-based reduction option 0: no range reduction based on marginals. 1: range reduction done based on marginals.</td>
<td>1</td>
</tr>
<tr>
<td>OBTTTDo</td>
<td>Optimality based tightening option 0: no range reduction based on optimality. 1: range reduction done based on optimality.</td>
<td>1</td>
</tr>
<tr>
<td>PDo</td>
<td>Number of probing problems allowed -2: automatically decided by BARON. -1: probing on all variables. 0: no range reduction by probing. n: probing on n variables.</td>
<td>-2</td>
</tr>
<tr>
<td>TDo</td>
<td>Nonlinear-feasibility-based range reduction option (poor man’s NLPs) 0: no bounds tightening is performed. 1: bounds tightening is performed.</td>
<td>1</td>
</tr>
</tbody>
</table>

#### 5.5.5.4 Tree management options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrPtStra</td>
<td>Branching point selection strategy 0: BARONs dynamic strategy 1: w-branching 2: bisection-branching 3: convex combination of 1 and 2</td>
<td>0</td>
</tr>
<tr>
<td>BrVarStra</td>
<td>Branching variable selection strategy 0: BARONs dynamic strategy 1: largest violation 2: longest edge</td>
<td>0</td>
</tr>
<tr>
<td>NodeSel</td>
<td>Specifies the node selection rule to be used for exploring the search tree. Specifies the node selection rule to be used for exploring the search tree. 0: BARONs mixed selection scheme 1: best bound 2: last in first out [LIFO] 3: minimum infeasibility</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 5.5.5.5 Local search options
### 5.5.5.6 Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocRes</td>
<td>Option to control output from local search</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: no local search output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: detailed results from local search will be printed to res.dat file</td>
<td></td>
</tr>
<tr>
<td>prfreq</td>
<td>Log output frequency in number of nodes</td>
<td>1000000</td>
</tr>
<tr>
<td>prlevel</td>
<td>Defines the level of log output printed.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: all log output is suppressed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: print log output</td>
<td></td>
</tr>
<tr>
<td>prtimefreq</td>
<td>Log output frequency in number of seconds</td>
<td>30</td>
</tr>
</tbody>
</table>

### 5.5.5.7 Subsolver Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowExternal</td>
<td>Indicator for use of External NLP solver with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of external GAMS NLP solver as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use the GAMS external NLP solver for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider the GAMS external NLP solver for local search</td>
<td></td>
</tr>
<tr>
<td>AllowFilterSD</td>
<td>Indicator for use of FILTERSD with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSD as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use FILTERSD for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider FILTERSD for local search</td>
<td></td>
</tr>
<tr>
<td>AllowFilterSQP</td>
<td>Indicator for use of FILTERSQP with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSQP as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use FILTERSQP for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider FILTERSQP for local search</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>AllowIpopt</td>
<td>Indicator for use of IPOPT with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>selectively permit or disallow the use of IPOPT as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Currently, this option defaults to 0 on Mac OS X.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use IPOPT for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider IPOPT for local search</td>
<td></td>
</tr>
<tr>
<td>AllowMinos</td>
<td>Indicator for use of MINOS with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>selectively permit or disallow the use of MINOS as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use MINOS for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider MINOS for local search</td>
<td></td>
</tr>
<tr>
<td>AllowSnopt</td>
<td>Indicator for use of SNOPT with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>selectively permit or disallow the use of SNOPT as an NLP subsolver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not use SNOPT for local search</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: consider SNOPT for local search</td>
<td></td>
</tr>
<tr>
<td>ExtNLPsolver</td>
<td>External GAMS NLP solver and option file (e.g. conopt.1)</td>
<td>conopt</td>
</tr>
<tr>
<td></td>
<td>Specifies the GAMS NLP solver to be used when NLPSol is set to -1 or 6. All</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GAMS NLP solvers are available through this option. If a non-existing solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is specified or the solver chosen cannot solve NLPs, NLPSol will be reset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to its default. A GAMS solver options file can be specified for the GAMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NLP solver by adding a dot followed by the options file number to the solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name, e.g., setting ExtNLPsolver to CONOPT.42 would instruct GAMS/CONOPT to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use options file conopt.o42.</td>
<td></td>
</tr>
<tr>
<td>LPAlg</td>
<td>Specifies the LP algorithm to be used (available only with CPLEX as the LP</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>solver)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: automatic selection of LP algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: primal simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: dual simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: barrier</td>
<td></td>
</tr>
<tr>
<td>LPSol</td>
<td>Specifies the LP/MIP Solver to be used</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>By default, BARON will select the LP solver and may switch between different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LP solvers during the search according to problem characteristics and solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance. The solvers CPLEX, if available, and CBC will be used for this</td>
<td></td>
</tr>
<tr>
<td></td>
<td>purpose. A single specific LP solver can be specified by setting this option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to a value other than the default. If the specified solver is not licensed,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BARON will default to automatic solver selection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1: Automatic LP solver selection and switching strategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: CPLEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7: XPRESS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8: CLP/CBC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15: HSL LA04</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>NLPSol</td>
<td>Specifies the NLP solver to be used</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>By default, BARON will select the NLP solver and may switch between</td>
<td></td>
</tr>
<tr>
<td></td>
<td>different NLP solvers during the search, based on problem characteristics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and solver performance. Any combination of licensed NLP solvers may</td>
<td></td>
</tr>
<tr>
<td></td>
<td>be used in that case. A single specific NLP solver can be specified by</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setting this option to a value other than the default. If the specified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>solver is not licensed, BARON will default to automatic solver selection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1: Automatic NLP solver selection and switching strategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Local search based on function evaluations alone with no calls to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>local solvers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: MINOS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: SNOPT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6: GAMS NLP solver (see ExtNLPsolver)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9: IPOPT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10: FILTERSD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14: FILTERSQP</td>
<td></td>
</tr>
</tbody>
</table>

### 5.5.5.8 Other Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompIIS</td>
<td>Request the computation of an Irreducible Inconsistent Set (IIS)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>In case of an infeasible problem, this option can be used to search for an IIS. Setting this option 1, works very well for most problems. 0: do not search for an IIS 1: the search for an IIS is based on a fast heuristic 2: an IIS is obtained using a deletion filtering algorithm 3: an IIS is obtained using an addition filtering algorithm 4: an IIS is obtained using an addition-deletion filtering algorithm 5: an IIS is obtained using a depth-first search algorithm</td>
<td></td>
</tr>
<tr>
<td>IISInt</td>
<td>Indicates whether general integers should be considered as potential members of the IIS When search for an IIS is requested through CompIIS, BARON assumes that the model is unlikely to include an error in terms of binaries, i.e., the binary definitions are assumed correct and the IIS output should be interpreted with respect to binary definitions. General integer bounds may be assumed as correct or can be questioned using the option IISint. Integrality is enforced in both cases. 0: do not consider general integers as part of an IIS, assume them to be correct 1: consider general integers (but not binaries) as part of an IIS</td>
<td>0</td>
</tr>
<tr>
<td>IISOrder</td>
<td>Order in which constraints are considered in the search for an IIS</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: auto set to aim for a small IIS depending on the value of CompIIS 1: arrange constraints in problem order 2: arrange constraints in ascending order of degree 3: arrange constraints in descending order of degree &gt;3: random order using IISorder as seed</td>
<td></td>
</tr>
<tr>
<td>InfBnd</td>
<td>infinity value to be used on bounds</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>If set to 0, then no bounds are used.</td>
<td></td>
</tr>
<tr>
<td>WantDual</td>
<td>whether to try to provide dual solution values</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: BARON may or may not return a dual solution. 1: BARON will return a dual solution using an inexpensive technique to solve a KKT system corresponding to the best primal solution identified</td>
<td></td>
</tr>
</tbody>
</table>
5.6 CBC

CBC (COIN-OR Branch and Cut) is an open-source mixed integer programming solver working with the COIN-OR LP solver CLP and the COIN-OR Cut generator library Cgl. The code has been written primarily by John J. Forrest. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

The CBC link in GAMS supports continuous, binary, integer, semicontinuous, semiinteger variables, special ordered sets of type 1 and 2, and branching priorities.

5.6.1 Usage

The following statement can be used inside your GAMS program to specify using CBC

```
Option LP = CBC;  { or MIP or RMIP }
```

The above statement should appear before the Solve statement. If CBC was specified as the default solver during GAMS installation, the above statement is not necessary.

For usage and syntax of solver options file, see Section The Solver Option File. Following is an example options file cbc.opt.
cuts root
perturbation off

It will cause CBC to use cut generators only in the root node and turns off the perturbation of the LP relaxation.

GAMS/CBC currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/CBC with BCH, please consider to use a GAMS system of version \leq 23.3.

The following GAMS parameters are currently supported by GAMS/CBC: reslim, iterlim, nodlim, optca, optcr, cheat, cutoff, and threads.

### 5.6.2 List of Options

There are many parameters which can affect the performance the CBCs Branch and Cut Algorithm. First just try with default settings and look carefully at the log file. Did cuts help? Did they take too long? Look at the output to see which cuts were effective and then do some tuning (see the option cuts). If the preprocessing reduced the size of the problem or strengthened many coefficients then it is probably wise to leave it on. Switch off heuristics which did not provide solutions. The other major area to look at is the search. Hopefully good solutions were obtained fairly early in the search so the important point is to select the best variable to branch on. See whether strong branching did a good job – or did it just take a lot of iterations? Adjust the options strongBranching and trustpseudocosts.

In the following, we summarize all available CBC options.

#### 5.6.2.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clocktype</td>
<td>type of clock for time measurement</td>
<td>wall</td>
</tr>
<tr>
<td>reslim</td>
<td>maximum seconds</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>special</td>
<td>options passed unseen to CBC</td>
<td></td>
</tr>
<tr>
<td>writemps</td>
<td>create MPS file for problem</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.6.2.2 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoScale</td>
<td>Whether to scale objective, rhs and bounds of problem if they</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>look odd (experimental)</td>
<td></td>
</tr>
<tr>
<td>biasLU</td>
<td>Whether factorization biased towards U</td>
<td>LX</td>
</tr>
<tr>
<td>bscale</td>
<td>Whether to scale in barrier (and ordering speed)</td>
<td>off</td>
</tr>
<tr>
<td>crash</td>
<td>Whether to create basis for problem</td>
<td>off</td>
</tr>
<tr>
<td>crossover</td>
<td>Whether to get a basic solution with the simplex algorithm after</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>the barrier algorithm finished</td>
<td></td>
</tr>
<tr>
<td>denseThreshold</td>
<td>Threshold for using dense factorization</td>
<td>-1</td>
</tr>
<tr>
<td>dualPivot</td>
<td>Dual pivot choice algorithm</td>
<td>automatic</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>factorization</td>
<td>Which factorization to use</td>
<td>normal</td>
</tr>
<tr>
<td>gamma</td>
<td>Whether to regularize barrier</td>
<td>off</td>
</tr>
<tr>
<td>idiotCrash</td>
<td>Whether to try idiot crash</td>
<td>-1</td>
</tr>
<tr>
<td>iterlim</td>
<td>Maximum number of iterations before stopping</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>KKT</td>
<td>Whether to use KKT factorization in barrier</td>
<td>0</td>
</tr>
<tr>
<td>maxFactor</td>
<td>Maximum number of iterations between refactorizations</td>
<td>200</td>
</tr>
<tr>
<td>passPresolve</td>
<td>How many passes in presolve</td>
<td>5</td>
</tr>
<tr>
<td>perturbation</td>
<td>Whether to perturb the problem</td>
<td>1</td>
</tr>
<tr>
<td>presolve</td>
<td>Whether to presolve problem</td>
<td>on</td>
</tr>
<tr>
<td>primalPivot</td>
<td>Primal pivot choice algorithm</td>
<td>automatic</td>
</tr>
<tr>
<td>primalWeight</td>
<td>Initially algorithm acts as if it costs this much to be infeasible</td>
<td>1e+10</td>
</tr>
<tr>
<td>psi</td>
<td>Two-dimension pricing factor for Positive Edge criterion</td>
<td>-0.5</td>
</tr>
<tr>
<td>randomSeedClp</td>
<td>Random seed for Clp</td>
<td>1234567</td>
</tr>
<tr>
<td>scaling</td>
<td>Whether to scale problem</td>
<td>automatic</td>
</tr>
<tr>
<td>smallFactorization</td>
<td>Threshold for using small factorization</td>
<td>-1</td>
</tr>
<tr>
<td>sparseFactor</td>
<td>Whether factorization treated as sparse</td>
<td>1</td>
</tr>
<tr>
<td>sprintCrash</td>
<td>Whether to try sprint crash</td>
<td>-1</td>
</tr>
<tr>
<td>startalg</td>
<td>LP solver for root node</td>
<td>dual</td>
</tr>
<tr>
<td>substitution</td>
<td>How long a column to substitute for in presolve</td>
<td>3</td>
</tr>
<tr>
<td>tol_dual</td>
<td>For an optimal solution no dual infeasibility may exceed this value</td>
<td>1e-07</td>
</tr>
<tr>
<td>tol_presolve</td>
<td>Tolerance to use in presolve</td>
<td>1e-08</td>
</tr>
<tr>
<td>tol_primal</td>
<td>For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value</td>
<td>1e-07</td>
</tr>
</tbody>
</table>

5.6.2.3 MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>costStrategy</td>
<td>How to use costs for branching priorities</td>
<td>off</td>
</tr>
<tr>
<td>cutoff</td>
<td>Bound on the objective value for all solutions</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td>cutoffConstraint</td>
<td>Whether to use cutoff as constraint</td>
<td>off</td>
</tr>
<tr>
<td>dump solutions</td>
<td>name of solutions index gdx file for writing alternate solutions</td>
<td></td>
</tr>
<tr>
<td>dump solutions merged</td>
<td>name of gdx file for writing all alternate solutions</td>
<td></td>
</tr>
<tr>
<td>expensiveStrong</td>
<td>Whether to do even more strong branching</td>
<td>0</td>
</tr>
<tr>
<td>extraVariables</td>
<td>Allow creation of extra integer variables</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>fixOnDj</td>
<td>Try heuristic based on fixing variables with reduced costs greater than this</td>
<td>-1</td>
</tr>
<tr>
<td>increment</td>
<td>A valid solution must be at least this much better than last integer solution</td>
<td>GAMS cheat</td>
</tr>
<tr>
<td>infeasibilityWeight</td>
<td>Each integer infeasibility is expected to cost this much</td>
<td>0</td>
</tr>
<tr>
<td>loglevel</td>
<td>amount of output printed by CBC</td>
<td>1</td>
</tr>
<tr>
<td>maxsol</td>
<td>Maximum number of solutions to save</td>
<td>100</td>
</tr>
<tr>
<td>mipstart</td>
<td>whether it should be tried to use the initial variable levels as initial MIP solution</td>
<td>0</td>
</tr>
<tr>
<td>multipleRootPasses</td>
<td>Do multiple root passes to collect cuts and solutions</td>
<td>0</td>
</tr>
<tr>
<td>nodeStrategy</td>
<td>What strategy to use to select the next node from the branch and cut tree</td>
<td>fewest</td>
</tr>
<tr>
<td>nodlim</td>
<td>node limit</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td>optca</td>
<td>Stop when gap between best possible and best less than this</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>optcr</td>
<td>Stop when gap between best possible and best known is less than this fraction of larger of two</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>OrbitalBranching</td>
<td>Whether to try orbital branching</td>
<td>off</td>
</tr>
<tr>
<td>parallelmode</td>
<td>whether to run opportunistic or deterministic</td>
<td>deterministic</td>
</tr>
<tr>
<td>preprocess</td>
<td>Whether to use integer preprocessing</td>
<td>sos</td>
</tr>
<tr>
<td>printfrequency</td>
<td>frequency of status prints</td>
<td>0</td>
</tr>
<tr>
<td>randomSeedCbc</td>
<td>Random seed for Cbc</td>
<td>-1</td>
</tr>
<tr>
<td>sollim</td>
<td>Maximum number of feasible solutions to get</td>
<td>∞</td>
</tr>
<tr>
<td>solvefinal</td>
<td>final solve of MIP with fixed discrete variables</td>
<td>1</td>
</tr>
<tr>
<td>solvetrace</td>
<td>name of trace file for solving information</td>
<td></td>
</tr>
<tr>
<td>solvetracenodefreq</td>
<td>frequency in number of nodes for writing to solve trace file</td>
<td>100</td>
</tr>
<tr>
<td>solvetracetimefreq</td>
<td>frequency in seconds for writing to solve trace file</td>
<td>5</td>
</tr>
<tr>
<td>sosPrioritize</td>
<td>How to deal with SOS priorities</td>
<td>off</td>
</tr>
<tr>
<td>strategy</td>
<td>Switches on groups of features</td>
<td>1</td>
</tr>
<tr>
<td>strongBranching</td>
<td>Number of variables to look at in strong branching</td>
<td>5</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads to try and use</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>tol_integer</td>
<td>For a feasible solution no integer variable may be more than this away from an integer value</td>
<td>1e-07</td>
</tr>
<tr>
<td>trustPseudoCosts</td>
<td>Number of branches before we trust pseudocosts</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.6.2.4 MIP Options for Cutting Plane Generators

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cliqueCuts</td>
<td>Whether to use Clique cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>conflictcuts</td>
<td>Conflict Cuts</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>cut.passes_root</td>
<td>Number of rounds that cut generators are applied in the root node</td>
<td>20 or 100</td>
</tr>
<tr>
<td>cut.passes_slow</td>
<td>Maximum number of rounds for slower cut generators</td>
<td>10</td>
</tr>
<tr>
<td>cut.passes_tree</td>
<td>Number of rounds that cut generators are applied in the tree</td>
<td>10</td>
</tr>
<tr>
<td>cutDepth</td>
<td>Depth in tree at which to do cuts</td>
<td>-1</td>
</tr>
<tr>
<td>cutLength</td>
<td>Length of a cut</td>
<td>-1</td>
</tr>
<tr>
<td>cuts</td>
<td>Switches all cut generators on or off</td>
<td>on</td>
</tr>
<tr>
<td>flowCoverCuts</td>
<td>Whether to use Flow Cover cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>gomoryCuts</td>
<td>Whether to use Gomory cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>gomorycuts2</td>
<td>Whether to use alternative Gomory cuts</td>
<td>off</td>
</tr>
<tr>
<td>knapsackCuts</td>
<td>Whether to use Knapsack cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>lagomoryCuts</td>
<td>Whether to use Lagrangean Gomory cuts</td>
<td>off</td>
</tr>
<tr>
<td>latwomirCuts</td>
<td>Whether to use Lagrangean TwoMir cuts</td>
<td>off</td>
</tr>
<tr>
<td>liftAndProjectCuts</td>
<td>Whether to use Lift and Project cuts</td>
<td>off</td>
</tr>
<tr>
<td>mirCuts</td>
<td>Whether to use Mixed Integer Rounding cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>probingCuts</td>
<td>Whether to use Probing cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>reduceAndSplitCuts</td>
<td>Whether to use Reduce-and-Split cuts</td>
<td>off</td>
</tr>
<tr>
<td>reduceAndSplitCuts2</td>
<td>Whether to use Reduce-and-Split cuts - style 2</td>
<td>off</td>
</tr>
<tr>
<td>residualCapacityCuts</td>
<td>Whether to use Residual Capacity cuts</td>
<td>off</td>
</tr>
<tr>
<td>twoMirCuts</td>
<td>Whether to use Two phase Mixed Integer Rounding cuts</td>
<td>root</td>
</tr>
<tr>
<td>zeroHalfCuts</td>
<td>Whether to use zero half cuts</td>
<td>ifmove</td>
</tr>
</tbody>
</table>

### 5.6.2.5 MIP Options for Primal Heuristics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine2Solutions</td>
<td>Whether to use crossover solution heuristic</td>
<td>off</td>
</tr>
<tr>
<td>combineSolutions</td>
<td>Whether to use combine solution heuristic</td>
<td>off</td>
</tr>
<tr>
<td>Dins</td>
<td>Whether to try Distance Induced Neighborhood Search</td>
<td>off</td>
</tr>
<tr>
<td>diveSolves</td>
<td>Diving solve option</td>
<td>100</td>
</tr>
<tr>
<td>DivingCoefficient</td>
<td>Whether to try Coefficient diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>DivingFractional</td>
<td>Whether to try Fractional diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>DivingGuided</td>
<td>Whether to try Guided diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>DivingLineSearch</td>
<td>Whether to try Linesearch diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>DivingPseudoCost</td>
<td>Whether to try Pseudocost diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>DivingRandom</td>
<td>Whether to try Diving heuristics</td>
<td>off</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>DivingVectorLength</td>
<td>Whether to try Vectorlength diving heuristic</td>
<td>off</td>
</tr>
<tr>
<td>dwHeuristic</td>
<td>Whether to try Dantzig Wolfe heuristic</td>
<td>off</td>
</tr>
<tr>
<td>feaspump</td>
<td>Whether to try the Feasibility Pump heuristic</td>
<td>on</td>
</tr>
<tr>
<td>feaspump_artcost</td>
<td>Costs ≥ this treated as artificials in feasibility pump</td>
<td>0</td>
</tr>
<tr>
<td>feaspump_cutoff</td>
<td>Fake cutoff for use in feasibility pump</td>
<td>0</td>
</tr>
<tr>
<td>feaspump_fracbab</td>
<td>Fraction in feasibility pump</td>
<td>0.5</td>
</tr>
<tr>
<td>feaspump_increment</td>
<td>Fake increment for use in feasibility pump</td>
<td>0</td>
</tr>
<tr>
<td>feaspump_passes</td>
<td>How many passes to do in the Feasibility Pump heuristic</td>
<td>20</td>
</tr>
<tr>
<td>greedyHeuristic</td>
<td>Whether to use a greedy heuristic</td>
<td>on</td>
</tr>
<tr>
<td>heuristics</td>
<td>Switches most primal heuristics on or off</td>
<td>1</td>
</tr>
<tr>
<td>hOptions</td>
<td>Heuristic options</td>
<td>0</td>
</tr>
<tr>
<td>localTreeSearch</td>
<td>Whether to use local tree search when a solution is found</td>
<td>0</td>
</tr>
<tr>
<td>naiveHeuristics</td>
<td>Whether to try some stupid heuristic</td>
<td>off</td>
</tr>
<tr>
<td>pivotAndComplement</td>
<td>Whether to try Pivot and Complement heuristic</td>
<td>off</td>
</tr>
<tr>
<td>pivotAndFix</td>
<td>Whether to try Pivot and Fix heuristic</td>
<td>off</td>
</tr>
<tr>
<td>proximitySearch</td>
<td>Whether to do proximity search heuristic</td>
<td>off</td>
</tr>
<tr>
<td>randomizedRounding</td>
<td>Whether to try randomized rounding heuristic</td>
<td>off</td>
</tr>
<tr>
<td>Rens</td>
<td>Whether to try Relaxation Enforced Neighborhood Search</td>
<td>off</td>
</tr>
<tr>
<td>Rins</td>
<td>Whether to try Relaxed Induced Neighborhood Search</td>
<td>off</td>
</tr>
<tr>
<td>roundingHeuristic</td>
<td>Whether to use simple (but effective) Rounding heuristic</td>
<td>on</td>
</tr>
<tr>
<td>VndVariableNeighborhoodSearch</td>
<td>Whether to try Variable Neighborhood Search</td>
<td>off</td>
</tr>
<tr>
<td>vubheuristic</td>
<td>Type of VUB heuristic</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.6.3 Detailed Options Description

In the following, we give a detailed description of all available CBC options.

**autoScale**: Whether to scale objective, rhs and bounds of problem if they look odd (experimental) ←

Range: boolean

Default: 0

**biasLU**: Whether factorization biased towards U ←

Range: UU, UX, LX, LL

Default: LX
bscale: Whether to scale in barrier (and ordering speed) ←

Range: off, on, off1, on1, off2, on2

Default: off

cliqueCuts: Whether to use Clique cuts ←

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglClique

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

clocktype: type of clock for time measurement ←

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>CPU clock</td>
</tr>
<tr>
<td>wall</td>
<td>Wall clock</td>
</tr>
</tbody>
</table>

Default: wall

combine2Solutions: Whether to use crossover solution heuristic ←

This heuristic does branch and cut on the problem given by fixing variables which have the same value in two or more solutions. It obviously only tries after two or more solutions. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: off

combineSolutions: Whether to use combine solution heuristic ←

This heuristic does branch and cut on given problem by just using variables which have appeared in one or more solutions. It is obviously only tried after two or more solutions have been found. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>onquick</td>
<td></td>
</tr>
<tr>
<td>bothquick</td>
<td></td>
</tr>
<tr>
<td>beforequick</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
conflictcuts: Conflict Cuts

Equivalent to setting cutoffconstraint=conflict

Range: boolean

Default: 0

costStrategy: How to use costs for branching priorities

Value 'priorities' assigns highest priority to variables with largest absolute cost. This primitive strategy can be surprisingly effective. Value 'columnorder' assigns the priorities 1, 2, 3, ... with respect to the column ordering. Value '01first' ('01last') assigns two sets of priorities such that binary variables get high (low) priority. Value 'length' assigns high priority to variables that occur in many equations.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>priorities</td>
<td></td>
</tr>
<tr>
<td>columnOrder</td>
<td></td>
</tr>
<tr>
<td>01first</td>
<td></td>
</tr>
<tr>
<td>binaryfirst</td>
<td>This is a deprecated setting. Please use 01first.</td>
</tr>
<tr>
<td>01last</td>
<td></td>
</tr>
<tr>
<td>binarylast</td>
<td>This is a deprecated setting. Please use 01last.</td>
</tr>
<tr>
<td>length</td>
<td></td>
</tr>
<tr>
<td>singletons</td>
<td></td>
</tr>
<tr>
<td>nonzero</td>
<td></td>
</tr>
<tr>
<td>generalForce</td>
<td></td>
</tr>
</tbody>
</table>

Default: off

crash: Whether to create basis for problem

If crash is set to 'on' and there is an all slack basis then Clp will flip or put structural variables into the basis with the aim of getting dual feasible. On average, dual simplex seems to perform better without it and there are alternative types of 'crash' for primal simplex, e.g. 'idiot' or 'sprint'. A variant due to Solow and Halim which is as 'on' but just flips is also available.

Range: off, on, solow_halim, lots, idiot1, idiot2, idiot3, idiot4, idiot5, idiot6, idiot7

Default: off

crossover: Whether to get a basic solution with the simplex algorithm after the barrier algorithm finished

Interior point algorithms do not obtain a basic solution. This option will crossover to a basic solution suitable for ranging or branch and cut.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>presolve</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>
cut\_passes\_root: Number of rounds that cut generators are applied in the root node

The default is to do 100 passes if the problem has less than 500 columns, 100 passes (but stop if the drop in the objective function value is small) if the problem has less than 5000 columns, and 20 passes otherwise. A negative value -n means that n passes are also applied if the objective does not drop.

Range: \{-\infty, ..., \infty\}

Default: 20 or 100

Synonym: passCuts

cut\_passes\_slow: Maximum number of rounds for slower cut generators

Some cut generators are fairly slow - this limits the number of times they are tried. The cut generators identified as 'may be slow' at present are Lift and project cuts and both versions of Reduce and Split cuts.

Range: \{-1, ..., \infty\}

Default: 10

Synonym: slowcutpasses

cut\_passes\_tree: Number of rounds that cut generators are applied in the tree

The default is to do one pass. A negative value -n means that n passes are also applied if the objective does not drop.

Range: \{-\infty, ..., \infty\}

Default: 10

Synonym: passTreeCuts

cutDepth: Depth in tree at which to do cuts

Cut generators may be off, on, on only at the root node, or on if they look useful. Setting this option to a positive value K let CBC call a cutgenerator on a node whenever the depth in the tree is a multiple of K. The default of -1 lets CBC decide.

Range: \{-1, ..., \infty\}

Default: -1

cutLength: Length of a cut

At present this only applies to Gomory cuts. -1 (default) leaves as is. Any value \geq 0 says that all cuts \leq this length can be generated both at root node and in tree. 0 says to use some dynamic lengths. If value \geq 10,000,000 then the length in tree is value\%10000000 - so 10000100 means unlimited length at root and 100 in tree.

Range: \{-1, ..., \infty\}

Default: -1

cutoff: Bound on the objective value for all solutions

All solutions must have a better objective value than the value of this option. CBC also updates this value whenever it obtains a solution to the value of the objective function of the solution minus the cutoff increment.

Range: real

Default: GAMS cutoff

cutoffConstraint: Whether to use cutoff as constraint

For some problems, cut generators and general branching work better if the problem would be infeasible if the cost is too high. If this option is enabled, the objective function is added as a constraint which right hand side is set to the current cutoff value (objective value of best known solution)
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>variable</td>
<td>Switches all cut generators on or off ←</td>
</tr>
<tr>
<td>forcevariable</td>
<td>CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior).</td>
</tr>
<tr>
<td>conflict</td>
<td>Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node.</td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**cuts:** Switches all cut generators on or off ←

This can be used to switch on or off all cut generators (apart from Reduce and Split). Then one can turn individual ones on or off. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Range: off, on, root, ifmove, forceOn

Default: on

**denseThreshold:** Threshold for using dense factorization ←

If processed problem ≤ this use dense factorization

Range: \{-1, ..., 10000\}

Default: -1

**Dins:** Whether to try Distance Induced Neighborhood Search ←

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>often</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**diveSolves:** Diving solve option ←
If >0 then do up to this many solves. However, the last digit is ignored and used for extra options: 1-3 enables fixing of satisfied integer variables (but not at bound), where 1 switches this off for that dive if the dive goes infeasible, and 2 switches it off permanently if the dive goes infeasible.

Range: {-1, ..., 200000}

Default: 100

**DivingCoefficient**: Whether to try Coefficient diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**DivingFractional**: Whether to try Fractional diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**DivingGuided**: Whether to try Guided diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>
Default: off

**DivingLineSearch**: Whether to try Linesearch diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**DivingPseudoCost**: Whether to try Pseudocost diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**DivingRandom**: Whether to try Diving heuristics

This switches on a random diving heuristic at various times. One may prefer to individually turn diving heuristics on or off. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

Synonym: DivingSome
DivingVectorLength: Whether to try Vectorlength diving heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

dualPivot: Dual pivot choice algorithm

The Dantzig method is simple but its use is deprecated. Steepest is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while automatic decides at each iteration based on information about the factorization. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also option psi.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>automatic</td>
<td></td>
</tr>
<tr>
<td>dantzig</td>
<td></td>
</tr>
<tr>
<td>partial</td>
<td></td>
</tr>
<tr>
<td>steepest</td>
<td></td>
</tr>
<tr>
<td>PEsteepest</td>
<td></td>
</tr>
<tr>
<td>PEDantzig</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td>Same as automatic. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: automatic

dumpsolutions: name of solutions index gdx file for writing alternate solutions

The name of a solutions index gdx file for writing alternate solutions found by CBC. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions.

Range: string

Default: empty

dumpsolutionsmerged: name of gdx file for writing all alternate solutions

Range: string

Default: empty

dwHeuristic: Whether to try Dantzig Wolfe heuristic
This heuristic is very very compute intensive. It tries to find a Dantzig Wolfe structure and use that. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before
Default: off

**expensiveStrong:** Whether to do even more strong branching

Strategy for extra strong branching. 0 is normal strong branching. 1, 2, 4, and 6 does strong branching on all fractional variables if at the root node (1), at depth less than modifier (2), objective equals best possible (4), or at depth less than modifier and objective equals best possible (6). 11, 12, 14, and 16 are like 1, 2, 4, and 6, respectively, but do strong branching on all integer (incl. non-fractional) variables. Values ≥ 100 are used to specify a depth limit (value/100), otherwise 5 is used. If the values ≥ 100, then above rules are applied to value%100.

Range: \{0, ..., \infty\}
Default: 0

**extraVariables:** Allow creation of extra integer variables

Switches on a trivial re-formulation that introduces extra integer variables to group together variables with same cost.

Range: \{-\infty, ..., \infty\}
Default: 0

**factorization:** Which factorization to use

The default is to use the normal CoinFactorization, but other choices are a dense one, OSL's, or one designed for small problems.

Range: normal, dense, simple, osl
Default: normal

**feaspump:** Whether to try the Feasibility Pump heuristic

This heuristic is due to Fischetti, Glover, and Lodi and uses a sequence of LPs to try and get an integer feasible solution. Some fine tuning is available by options passFeasibilityPump and pumpTune. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Default: on

Synonym: feasibilityPump

**feaspump_artcost:** Costs $\geq$ this treated as artificials in feasibility pump

A value of 0.0 means off. Otherwise, variables with costs $\geq$ this are treated as artificial variables and fixed to lower bound in feasibility pump.

Range: $[0, \infty]$  
Default: 0  

Synonym: artificialCost

**feaspump_cutoff:** Fake cutoff for use in feasibility pump

A value of 0.0 means off. Otherwise, add a constraint forcing objective below this value in feasibility pump

Range: real  
Default: 0  

Synonym: pumpCutoff

**feaspump_fracbab:** Fraction in feasibility pump

After a pass in the feasibility pump, variables which have not moved about are fixed and if the preprocessed model is smaller than this fraction of the original problem, a few nodes of branch and bound are done on the reduced problem.

Range: $[1e-05, 1.1]$  
Default: 0.5  

Synonym: fractionforBAB

**feaspump_increment:** Fake increment for use in feasibility pump

A value of 0.0 means off. Otherwise use as absolute increment to cutoff when solution found in feasibility pump

Range: real  
Default: 0  

Synonym: pumpIncrement

**feaspump_passes:** How many passes to do in the Feasibility Pump heuristic

Range: $\{0, ..., 10000\}$  
Default: 20  

Synonym: passFeasibilityPump

**fixOnDj:** Try heuristic based on fixing variables with reduced costs greater than this

If this is set integer variables with reduced costs greater than this will be fixed before branch
and bound - use with extreme caution!

Range: real

Default: -1

**flowCoverCuts**: Whether to use Flow Cover cuts

- Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see
  cutDepth on how to fine tune the behavior).
- Value 'root' lets CBC run the cut generator generate only at the root node.
- Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good
  and moves the objective value.
- Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Reference: [https://github.com/coin-or/Cgl/wiki/CglFlowCover](https://github.com/coin-or/Cgl/wiki/CglFlowCover)

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

**gamma**: Whether to regularize barrier

Range: off, on, gamma, delta, onstrong, gammastrong, deltastrong

Default: off

**gomoryCuts**: Whether to use Gomory cuts

The original cuts - beware of imitations! Having gone out of favor, they are now more
fashionable as LP solvers are more robust and they interact well with other cuts. They
will almost always give cuts (although in this executable they are limited as to number of
variables in cut). However the cuts may be dense so it is worth experimenting (Long allows
any length).

- Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see
  cutDepth on how to fine tune the behavior).
- Value 'root' lets CBC run the cut generator generate only at the root node.
- Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good
  and moves the objective value.
- Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Reference: [https://github.com/coin-or/Cgl/wiki/CglGomory](https://github.com/coin-or/Cgl/wiki/CglGomory)

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal, forceLongOn, long

Default: ifmove

**gomorycuts2**: Whether to use alternative Gomory cuts

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see
cutDepth on how to fine tune the behavior).

Value 'root' lets CBC run the cut generator generate only at the root node.

Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good
and moves the objective value.

Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

This version is by Giacomo Nannicini and may be more robust than gomoryCuts.

Range: off, on, root, ifmove, forceOn, endonly, long, longroot, longifmove, forceLongOn, longendonly

Default: off

**Synonym**: GMICuts
**greedyHeuristic**: Whether to use a greedy heuristic

This heuristic tries to obtain a feasible solution by just fixing a percentage of variables and then try a small branch and cut run. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: on

**heuristics**: Switches most primal heuristics on or off

This option can be used to switch on or off all heuristics that search for feasible solutions, except for the local tree search, as it dramatically alters the search. Then individual heuristics can be turned off or on.

Range: boolean

Default: 1

Synonym: heuristicsOnOff

**hOptions**: Heuristic options

Value 1 stops heuristics immediately if the allowable gap has been reached. Other values are for the feasibility pump - 2 says do exact number of passes given, 4 only applies if an initial cutoff has been given and says relax after 50 passes, while 8 will adapt the cutoff rhs after the first solution if it looks as if the code is stalling.

Range: {-∞, ..., ∞}

Default: 0

**idiotCrash**: Whether to try idiot crash

This is a type of 'crash' which works well on some homogeneous problems. It works best on problems with unit elements and rhs but will do something to any model. It should only be used before the primal simplex algorithm. It can be set to -1 when the code decides for itself whether to use it, 0 to switch off, or n > 0 to do n passes.

Range: {-1, ..., ∞}

Default: -1

**increment**: A valid solution must be at least this much better than last integer solution

Whenever a solution is found the bound on the objective value for new solutions is set to the objective function of the found solution (in a minimization sense) plus this. If it is not set then CBC will try and work one out, e.g. if all objective coefficients are multiples of 0.01 and only integer variables have entries in the objective function, then the increment can be set to 0.01. Be careful if setting this to a negative value!

Range: real

Default: GAMS cheat

**infeasibilityWeight**: Each integer infeasibility is expected to cost this much
A primitive way of deciding which node to explore next. Satisfying each integer infeasibility is expected to cost this much.

Range: $[0, \infty]$

Default: 0

**iterlim**: Maximum number of iterations before stopping

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

Range: $\{0, \ldots, \infty\}$

Default: GAMS iterlim

Synonym: maxIterations

**KKT**: Whether to use KKT factorization in barrier

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: [https://github.com/coin-or/Cgl/wiki/CglKnapsackCover](https://github.com/coin-or/Cgl/wiki/CglKnapsackCover)

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal

Default: ifmove

**knapsackCuts**: Whether to use Knapsack cuts

This is a gross simplification of 'A Relax-and-Cut Framework for Gomory's Mixed-Integer Cuts' by Matteo Fischetti & Domenico Salvagnin. This simplification just uses original constraints while modifying objective using other cuts. So you don't use messy constraints generated by Gomory etc. A variant is to allow non messy cuts e.g. clique cuts. So 'only' does this while 'clean' also allows integral valued cuts. 'End' is recommended and waits until other cuts have finished before it does a few passes. The length options for gomory cuts are used.

Range: off, endonlyroot, endcleanroot, root, endonly, endclean, endboth, onlyaswell, cleanaswell, bothaswell, onlyinstead, cleaninstead, bothinstead, onlyaswellroot, cleanaswellroot, bothaswellroot

Default: off

**lagomoryCuts**: Whether to use Lagrangean Gomory cuts

This is a gross simplification of 'A Relax-and-Cut Framework for Gomory's Mixed-Integer Cuts' by Matteo Fischetti & Domenico Salvagnin. This simplification just uses original constraints while modifying objective using other cuts. So you don't use messy constraints generated by Gomory etc. A variant is to allow non messy cuts e.g. clique cuts. So 'only' does this while 'clean' also allows integral valued cuts. 'End' is recommended and waits until other cuts have finished before it does a few passes. The length options for gomory cuts are used.

Range: off, endonlyroot, endcleanroot, root, endonly, endclean, endboth, onlyaswell, cleanaswell, bothaswell, onlyinstead, cleaninstead, bothinstead, onlyaswellroot, cleanaswellroot, bothaswellroot

Default: off

**latwomirCuts**: Whether to use Lagrangean TwoMir cuts

This is a Lagrangean relaxation for TwoMir cuts. See lagomoryCuts for description of options.

Range: off, endonlyroot, endcleanroot, endbothroot, endonly, endclean, endboth, onlyaswell, cleanaswell, bothaswell, onlyinstead, cleaninstead, bothinstead

Default: off
**liftAndProjectCuts**: Whether to use Lift and Project cuts

These cuts may be expensive to compute. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: [https://github.com/coin-or/Cgl/wiki/CglLandP](https://github.com/coin-or/Cgl/wiki/CglLandP)

- Range: off, on, root, ifmove, forceOn
- Default: off

**localTreeSearch**: Whether to use local tree search when a solution is found

The heuristic is from Fischetti and Lodi and is not really a heuristic although it can be used as one (with limited functionality). It is not switched on when heuristics are switched on.

- Range: boolean
- Default: 0

**loglevel**: amount of output printed by CBC

- Range: \(\{0, \ldots, \infty\}\)
- Default: 1

**maxFactor**: Maximum number of iterations between refactorizations

If this is left at its default value of 200 then CLP will guess a value to use. CLP may decide to re-factorize earlier for accuracy.

- Range: \(\{1, \ldots, \infty\}\)
- Default: 200

**maxsol**: Maximum number of solutions to save

Maximal number of solutions to store during search and to dump into gdx files if dumpSolutions options is set.

- Range: \(\{0, \ldots, \infty\}\)
- Default: 100
- Synonym: maxSavedSolutions

**mipstart**: whether it should be tried to use the initial variable levels as initial MIP solution

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

- Range: boolean
- Default: 0

**mirCuts**: Whether to use Mixed Integer Rounding cuts
Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglMixedIntegerRounding2

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

Synonym: mixedIntegerRoundingCuts

**multipleRootPasses:** Do multiple root passes to collect cuts and solutions

Solve (in parallel, if enabled) the root phase this number of times, each with its own different seed, and collect all solutions and cuts generated. The actual format is aabbcc where aa is the number of extra passes; if bb is non zero, then it is number of threads to use (otherwise uses threads setting); and cc is the number of times to do root phase. The solvers do not interact with each other. However if extra passes are specified then cuts are collected and used in later passes - so there is interaction there. Some parts of this implementation have their origin in idea of Andrea Lodi, Matteo Fischetti, Michele Monaci, Domenico Salvagnin, and Andrea Tramontani.

Range: \( \{0, \ldots, \infty\} \)

Default: 0

**naiveHeuristics:** Whether to try some stupid heuristic

This is naive heuristics which, e.g., fix all integers with costs to zero! Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**nodeStrategy:** What strategy to use to select the next node from the branch and cut tree

Normally before a feasible solution is found, CBC will choose a node with fewest infeasibilities. Alternatively, one may choose tree-depth as the criterion. This requires the minimal amount of memory, but may take a long time to find the best solution. Additionally, one may specify whether up or down branches must be selected first (the up-down choice will carry on after a first solution has been bound). The default choice 'hybrid' does breadth first on small depth nodes and then switches to 'fewest'.

Range: hybrid, fewest, depth, upfewest, downfewest, updepth, downdepth

Default: fewest
nodlim: node limit

Maximum number of nodes that are enumerated in the Branch and Bound tree search.

Range: \{-1, ..., \infty\}

Default: GAMS nodlim

Synonyms: maxNodes nodelim

optca: Stop when gap between best possible and best less than this

If the gap between best known solution and the best possible solution is less than this value, then the search will be terminated. Also see ratioGap.

Range: \([0, \infty]\)

Default: GAMS optca

Synonym: allowableGap

optcr: Stop when gap between best possible and best known is less than this fraction of larger of two

If the gap between the best known solution and the best possible solution is less than this fraction of the objective value at the root node then the search will terminate. See 'allowableGap' for a way of using absolute value rather than fraction.

Range: \([0, \infty]\)

Default: GAMS optcr

Synonym: ratioGap

OrbitalBranching: Whether to try orbital branching

This switches on Orbital branching. Value 'on' just adds orbital, 'strong' tries extra fixing in strong branching.

Range: off, on, strong, force, simple

Default: off

parallelmode: whether to run opportunistic or deterministic

Determines whether a parallel MIP search (threads > 1) should be done in a deterministic (i.e., reproducible) way or in a possibly faster but not necessarily reproducible way

Range: opportunistic, deterministic

Default: deterministic

passPresolve: How many passes in presolve

Normally Presolve does 10 passes but you may want to do less to make it more lightweight or do more if improvements are still being made. As Presolve will return if nothing is being taken out, you should not normally need to use this fine tuning.

Range: \{-200, ..., 100\}

Default: 5
perturbation: Whether to perturb the problem

Perturbation helps to stop cycling, but CLP uses other measures for this. However, large problems and especially ones with unit elements and unit right hand sides or costs benefit from perturbation. Normally CLP tries to be intelligent, but one can switch this off.

Range: boolean

Default: 1

pivotAndComplement: Whether to try Pivot and Complement heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: off

pivotAndFix: Whether to try Pivot and Fix heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Default: off

preprocess: Whether to use integer preprocessing

This tries to reduce size of model in a similar way to presolve and it also tries to strengthen the model - this can be very useful and is worth trying. Value 'save' saves the presolved problem to a file presolved.mps. Value 'equal' will turn inequality-cliques into equalities. Value 'sos' lets CBC search for rows with upper bound 1 and where all nonzero coefficients are 1 and creates special ordered sets if the sets are not overlapping and all integer variables (except for at most one) are in the sets. Value 'trysos' is same as 'sos', but allows any number of integer variables outside of sets. Value 'equalall' lets CBC turn all valid inequalities into equalities by adding integer slack variables.

Range: off, on, save, equal, sos, trysos, equalall, strategy, aggregate, force esos, stopaftersaving

Default: sos

presolve: Whether to presolve problem

Presolve analyzes the model to find such things as redundant equations, equations which fix some variables, equations which can be transformed into bounds, etc. For the initial solve of any problem this is worth doing unless one knows that it will have no effect. Option 'on' will normally do 5 passes, while using 'more' will do 10.
5.6 CBC

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>off</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>more</td>
<td></td>
</tr>
</tbody>
</table>

Default: on

**primalPivot**: Primal pivot choice algorithm

The Dantzig method is simple but its use is deprecated. Exact devex is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while 'change' initially does 'dantzig' until the factorization becomes denser. This is still a work in progress. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also Towhid, M., Desrosiers, J., Soumis, F., The positive edge criterion within COIN-OR’s CLP; Omer, J., Towhid, M., Soumis, F., The positive edge pricing rule for the dual simplex.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>automatic</td>
<td>Same as automatic. This is a deprecated setting.</td>
</tr>
<tr>
<td>exact</td>
<td></td>
</tr>
<tr>
<td>dantzig</td>
<td></td>
</tr>
<tr>
<td>partial</td>
<td></td>
</tr>
<tr>
<td>steepest</td>
<td></td>
</tr>
<tr>
<td>change</td>
<td></td>
</tr>
<tr>
<td>sprint</td>
<td></td>
</tr>
<tr>
<td>PEsteep</td>
<td></td>
</tr>
<tr>
<td>PEdantzig</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td></td>
</tr>
</tbody>
</table>

Default: automatic

**primalWeight**: Initially algorithm acts as if it costs this much to be infeasible

The primal algorithm in Clp is a single phase algorithm as opposed to a two phase algorithm where you first get feasible then optimal. So Clp is minimizing this weight times the sum of primal infeasibilities plus the true objective function (in minimization sense). Too high a value may mean more iterations, while too low a value means the algorithm may iterate into the wrong directory for long and then has to increase the weight in order to get feasible.

Range: [1e-20, ∞]

Default: 1e+10

**printfrequency**: frequency of status prints

Controls the number of nodes that are evaluated between status prints.

Range: {0, ..., ∞}

Default: 0
**probingCuts**: Whether to use Probing cuts

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Value 'forceOnBut' turns on probing and forces CBC to do probing at every node, but does only probing, not strengthening etc. Value 'strong' forces CBC to strongly do probing at every node, that is, also when CBC would usually turn it off because it hasn't found something. Value 'forceOnButStrong' is like 'forceOnStrong', but does only probing (column fixing) and turns off row strengthening, so the matrix will not change inside the branch and bound. Reference: [https://github.com/coin-or/Cgl/wiki/CglProbing](https://github.com/coin-or/Cgl/wiki/CglProbing)

Range: off, on, root, ifmove, forceOn, onglobal, forceonglobal, forceOnBut, forceOnStrong, forceOnButStrong, strongRoot

Default: ifmove

**proximitySearch**: Whether to do proximity search heuristic

This heuristic looks for a solution close to the incumbent solution (Fischetti and Monaci, 2012). The idea is to define a sub-MIP without additional constraints but with a modified objective function intended to attract the search in the proximity of the incumbent. The approach works well for 0-1 MIPs whose solution landscape is not too irregular (meaning the there is reasonable probability of finding an improved solution by flipping a small number of binary variables), in particular when it is applied to the first heuristic solutions found at the root node. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Default: off

**psi**: Two-dimension pricing factor for Positive Edge criterion

The Positive Edge criterion has been added to select incoming variables to try and avoid degenerate moves. Variables not in the promising set have their infeasibility weight multiplied by psi, so 0.01 would mean that if there were any promising variables, then they would always be chosen, while 1.0 effectively switches the algorithm off. There are two ways of switching this feature on. One way is to set psi to a positive value and then the Positive Edge criterion will be used for both primal and dual simplex. The other way is to select PEsteepest in dualpivot choice (for example), then the absolute value of psi is used. Code donated by Jeremy Omer. See Towhidi, M., Desrosiers, J., Soumis, F., The positive edge criterion within COIN-OR’s CLP; Omer, J., Towhidi, M., Soumis, F., The positive edge pricing rule for the dual simplex.

Range: [-1.1, 1.1]

Default: -0.5
randomizedRounding: Whether to try randomized rounding heuristic ←

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Default: off

randomSeedCbc: Random seed for Cbc ←

Allows initialization of the random seed for pseudo-random numbers used in heuristics such as the Feasibility Pump to decide whether to round up or down. The special value of 0 lets Cbc use the time of the day for the initial seed.

Range: \{-1, ..., \infty\}

Default: -1

Synonym: randomCbcSeed

randomSeedClp: Random seed for Clp ←

Initialization of the random seed for pseudo-random numbers used to break ties in degenerate problems. This may yield a different continuous optimum and, in the context of Cbc, different cuts and heuristic solutions. The special value of 0 lets CLP use the time of the day for the initial seed.

Range: \{0, ..., \infty\}

Default: 1234567

Synonym: randomSeed

reduceAndSplitCuts: Whether to use Reduce-and-Split cuts ←

These cuts may be expensive to generate. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Reference: https://github.com/coin-or/Cgl/wiki/CglRedSplit

Range: off, on, root, ifmove, forceOn

Default: off

reduceAndSplitCuts2: Whether to use Reduce-and-Split cuts - style 2 ←
This switches on reduce and split cuts (either at root or in entire tree). This version is by Giacomo Nannicini based on Francois Margot’s version. Standard setting only uses rows in tableau $\leq 256$, long uses all. These cuts may be expensive to generate. See option cuts for more information on the possible values.

Range: off, on, root, longOn, longRoot

Default: off

Synonym: reduce2AndSplitCuts

**Rens**: Whether to try Relaxation Enforced Neighborhood Search $\leftrightarrow$

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve. Value 'on' just does 50 nodes. 200, 1000, and 10000 does that many nodes.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>dj</td>
<td></td>
</tr>
<tr>
<td>djbefore</td>
<td></td>
</tr>
<tr>
<td>usesolution</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>0</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>1</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: off

**residualCapacityCuts**: Whether to use Residual Capacity cuts $\leftrightarrow$

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglResidualCapacity

Range: off, on, root, ifmove, forceOn

Default: off

**reslim**: maximum seconds $\leftrightarrow$

Range: $[-1, \infty]$  
Default: GAMS reslim  
Synonym: seconds

**Rins**: Whether to try Relaxed Induced Neighborhood Search $\leftrightarrow$

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.
5.6 CBC

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
<tr>
<td>often</td>
<td></td>
</tr>
</tbody>
</table>

Default: off

**roundingHeuristic**: Whether to use simple (but effective) Rounding heuristic

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>Same as off. This is a deprecated setting.</td>
</tr>
<tr>
<td>on</td>
<td>Same as on. This is a deprecated setting.</td>
</tr>
<tr>
<td>both</td>
<td></td>
</tr>
<tr>
<td>before</td>
<td></td>
</tr>
</tbody>
</table>

Default: on

**scaling**: Whether to scale problem

Scaling can help in solving problems which might otherwise fail because of lack of accuracy. It can also reduce the number of iterations. It is not applied if the range of elements is small. When the solution is evaluated in the unscaled problem, it is possible that small primal and/or dual infeasibilities occur. Option 'equilibrium' uses the largest element for scaling. Option 'geometric' uses the squareroot of the product of largest and smallest element. Option 'auto' let CLP choose a method that gives the best ratio of the largest element to the smallest one.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>equilibrium</td>
<td></td>
</tr>
<tr>
<td>geometric</td>
<td></td>
</tr>
<tr>
<td>automatic</td>
<td></td>
</tr>
<tr>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>rowsonly</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td>Same as automatic. This is a deprecated setting.</td>
</tr>
</tbody>
</table>

Default: automatic

**smallFactorization**: Threshold for using small factorization

If processed problem ≤ this use small factorization
Range: \{-1, ..., 10000\}
Default: -1

**sollim:** Maximum number of feasible solutions to get \(\rightarrow\)

Range: \(\{1, ..., \infty\}\)
Default: \(\infty\)
Synonym: maxSolutions

**solvefinal:** final solve of MIP with fixed discrete variables \(\leftrightarrow\)

Whether the MIP with discrete variables fixed to solution values should be solved after CBC finished.
Range: boolean
Default: 1

**solvetrace:** name of trace file for solving information \(\leftrightarrow\)

Name of file for writing solving progress information during solve.
Range: string
Default: *empty*

**solvetracenodefreq:** frequency in number of nodes for writing to solve trace file \(\leftrightarrow\)

Range: \(\{0, ..., \infty\}\)
Default: 100

**solvetracetimefreq:** frequency in seconds for writing to solve trace file \(\leftrightarrow\)

Range: \([0, \infty]\]
Default: 5

**sosPrioritize:** How to deal with SOS priorities \(\leftrightarrow\)

This sets priorities for SOS. Values 'high' and 'low' just set a priority relative to the for integer variables. Value 'orderhigh' gives first highest priority to the first SOS and integer variables a low priority. Value 'orderlow' gives integer variables a high priority then SOS in order.

Range: off, high, low, orderhigh, orderlow
Default: off

**sparseFactor:** Whether factorization treated as sparse \(\leftrightarrow\)

Range: boolean
Default: 1

**special:** options passed unseen to CBC \(\leftrightarrow\)
This parameter lets you specify CBC options which are not supported by the GAMS/CBC interface. The string value given to this parameter is split up into parts at each space and added to the array of parameters given to CBC (in front of the -solve command). Hence, you can use it like the command line parameters for the CBC standalone version.

Range: string

Default: empty

**sprintCrash**: Whether to try sprint crash

For long and thin problems this method may solve a series of small problems created by taking a subset of the columns. The idea as 'Sprint' was introduced by J. Forrest after an LP code of that name of the 60's which tried the same tactic (not totally successfully). CPLEX calls it 'sifting'. -1 lets CLP automatically choose the number of passes, 0 is off, n is number of passes

Range: {-1, ..., ∞}

Default: -1

Synonym: sifting

**startalg**: LP solver for root node

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>primal</td>
<td>Primal Simplex algorithm</td>
</tr>
<tr>
<td>dual</td>
<td>Dual Simplex algorithm</td>
</tr>
<tr>
<td>barrier</td>
<td>Primal-dual predictor-corrector algorithm</td>
</tr>
</tbody>
</table>

Default: dual

**strategy**: Switches on groups of features

This turns on newer features. Use 0 for easy problems, 1 is default, 2 is aggressive. 1 uses Gomory cuts with a tolerance of 0.01 at the root node, does a possible restart after 100 nodes if many variables could be fixed, activates a diving and RINS heuristic, and makes the feasibility pump more aggressive.

Range: {0, ..., 2}

Default: 1

**strongBranching**: Number of variables to look at in strong branching

In order to decide which variable to branch on, the code will choose up to this number of unsatisfied variables to try minimal up and down branches on. Then the most effective one is chosen. If a variable is branched on many times then the previous average up and down costs may be used - see also option trustPseudoCosts.

Range: {0, ..., ∞}

Default: 5

**substitution**: How long a column to substitute for in presolve
Normally Presolve gets rid of 'free' variables when there are no more than 3 coefficients in a row. If you increase this, the number of rows may decrease but the number of coefficients may increase.

Range: \([0, ..., 10000]\)

Default: 3

**threads**: Number of threads to try and use

Range: \([1, ..., 99]\)

Default: GAMS threads

**tol_dual**: For an optimal solution no dual infeasibility may exceed this value

Normally the default tolerance is fine, but one may want to increase it a bit if the dual simplex algorithm seems to be having a hard time. One method which can be faster is to use a large tolerance e.g. 1.0e-4 and the dual simplex algorithm and then to clean up the problem using the primal simplex algorithm with the correct tolerance (remembering to switch off presolve for this final short clean up phase).

Range: \([1e-20, \infty]\)

Default: 1e-07

Synonym: dualTolerance

**tol_integer**: For a feasible solution no integer variable may be more than this away from an integer value

Beware of setting this smaller than the primal feasibility tolerance.

Range: \([1e-20, 0.5]\)

Default: 1e-07

Synonym: integerTolerance

**tol_presolve**: Tolerance to use in presolve

One may want to increase this tolerance if presolve says the problem is infeasible and one has awkward numbers and is sure that the problem is really feasible.

Range: \([1e-20, \infty]\)

Default: 1e-08

Synonym: preTolerance

**tol_primal**: For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value

Normally the default tolerance is fine, but one may want to increase it a bit if the primal simplex algorithm seems to be having a hard time.

Range: \([1e-20, \infty]\)

Default: 1e-07

Synonym: primalTolerance
**trustPseudoCosts**: Number of branches before we trust pseudocosts 🔄

Using strong branching computes pseudo-costs. This parameter determines after how many branches for a variable we just trust the pseudo costs and do not do any more strong branching.

Range: \(-3, \ldots, \infty\)

Default: 10

**twoMirCuts**: Whether to use Two phase Mixed Integer Rounding cuts 🔄

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: [https://github.com/coin-or/Cgl/wiki/CglTwomir](https://github.com/coin-or/Cgl/wiki/CglTwomir)

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal, forceLongOn

Default: root

**VndVariableNeighborhoodSearch**: Whether to try Variable Neighborhood Search 🔄

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before, intree

Default: off

**vubheuristic**: Type of VUB heuristic 🔄

This heuristic tries and fix some integer variables.

Range: \{-2, \ldots, 20\}

Default: -1

**writemps**: create MPS file for problem 🔄

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

Range: string

Default: empty

**zeroHalfCuts**: Whether to use zero half cuts 🔄

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. This implementation was written by Alberto Caprara.

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove
5.7 CONOPT

Author
Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.7.1 Introduction

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there is a large number of different solvers available and the number is growing.

The most important distinction between the solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution (so called Global Solvers) can usually not solve very large models. As a contrast most Local Solvers can work with much larger models, and models with over 10,000 variables and constraints are not unusual. If the model has the right mathematical properties, e.g. is convex, then Local Solvers will find a global optimum. Unfortunately, the mathematical machinery for testing whether a general NLP model is convex or not has not yet been developed (and is expected to be in the class or hard problems).

It is almost impossible to predict how difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot select the best algorithm for you automatically. When GAMS is installed you must select one of the nonlinear programming algorithms as the default solver for NLP models. If you want to switch between algorithms for a particular model you may add the statement

Option NLP = <solvername>, in your GAMS source file before the Solve statement, you may add NLP = <solvername> on the GAMS command line or by rerunning the gamsinst program.

The only reliable way to find which solver to use for a particular class of models is so far to experiment. However, there are a few rules of thumb:

GAMS/CONOPT is well suited for models with very nonlinear constraints. If you experience that a solver has problems maintaining feasibility during the optimization you should try CONOPT. On the other hand, if you have a model with few nonlinearities outside the objective function then other solvers could be the best solver.

GAMS/CONOPT has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom. If you have a model with roughly the same number of constraints as variable you should try CONOPT. CONOPT can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System.

GAMS/CONOPT can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT. IPOPT and KNITRO will also use second derivatives, but the method is very different and it is not possible to predict which solver will be better.

GAMS/CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one then CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective terms are eliminated from the model and the constraints are moved into the objective function.

GAMS/CONOPT has many built-in tests and messages, and many models that can and should be improved by the modeler are rejected with a constructive message. CONOPT is therefore also a helpful debugging tool during model development. The best solver for the final, debugged model may or may not be CONOPT.
GAMS/CONOPT has been designed for large and sparse models. This means that both the number of variables and equations can be large. Indeed, NLP models with over 100,000 equations and variables have been solved successfully, and CNS models with over 1,000,000 equations and variables have also been solved. The components used to build CONOPT have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT can also be used for denser models, but the performance will suffer significantly.

GAMS/CONOPT is designed for models with smooth functions, but it can also be applied to models that do not have differentiable functions, in GAMS called DNLP models. However, CONOPT will use the same algorithm used for a real NLP model and it will search for a point that satisfies standard first-order optimality conditions without taking into account that parts of the model could be non-smooth or non-differentiable. The lack of smoothness may confuse the algorithm in CONOPT causing slow convergence, and a point that satisfies standard first-order optimality conditions may not even exist. There are therefore no guarantees whatsoever for this class of models. If CONOPT terminates with a locally optimal solution then the solution will indeed be locally optimal. However, you will sometimes get termination messages like "Convergence too slow" or "No change in objective although the reduced gradient is greater than the tolerance" that indicate unsuccessful termination. The final point may or may not be locally optimal. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section NLP and DNLP Models.

Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT has considerable build-in logic that selects a solution approach that seems to be best suited for the type of model at hand, and the approach is adjusted dynamically as information about the behavior of the model is collected and updated. The description of the CONOPT algorithm has therefore been moved to an appendix (Appendix A) and most modelers can skip it. However, if you are solving very large or complex models or if you are experiencing solution difficulties you may benefit from using non-standard tolerances or options, in which case you will need some understanding of what CONOPT is doing to your model. Some guidelines for selecting options can be found at the end of Appendix A and a list of all options and tolerances is shown in Appendix B.

The main text of this User's Guide will give a short overview over the iteration output you will see on the screen (section Iteration Output), and explain the termination messages (section GAMS/CONOPT Termination Messages). We will then discuss function evaluation errors (section Function Evaluation Errors), the use of options (section The CONOPT Options File), and give a CONOPT perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section Hints on Good Model Formulation). Finally, we will discuss the difference between NLP and DNLP models (section NLP and DNLP Models).

5.7.2 Iteration Output

On most machines you will by default get a logline on your screen or terminal at regular intervals. The iteration log may look something like this:

```
CONOPT 3 Jul 4, 2012 23.9.4 WEX 35892.35906 WEI x86_64/MS Windows

C O N O P T 3 version 3.15G
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark

Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
0 0 1.6354151782E+01 (Input point)

Pre-triangular equations: 2
Post-triangular equations: 1
```
<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>0</td>
<td>33</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>0</td>
<td>43</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>53</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>0</td>
<td>63</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>73</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>83</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>93</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>0</td>
<td>103</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>113</td>
<td>2.1776589484E+00</td>
<td>0.0E+00</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>119</td>
<td>8.7534351971E-01</td>
<td>0.0E+00</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Feasible solution. Value of objective = 1.00525015566

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Objective</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>226</td>
<td>3</td>
<td>1</td>
<td>1.0092586645E+00</td>
<td>4.4E-04</td>
<td>38</td>
<td>1.0E+00</td>
<td>3</td>
<td>T</td>
</tr>
<tr>
<td>231</td>
<td>3</td>
<td>1</td>
<td>1.0121749760E+00</td>
<td>1.4E+00</td>
<td>24</td>
<td>4.8E-01</td>
<td>9</td>
<td>T</td>
</tr>
<tr>
<td>236</td>
<td>3</td>
<td>1</td>
<td>1.0128148550E+00</td>
<td>4.8E-06</td>
<td>13</td>
<td>5.8E-02</td>
<td>12</td>
<td>F</td>
</tr>
<tr>
<td>241</td>
<td>3</td>
<td>1</td>
<td>1.0128161551E+00</td>
<td>2.5E-06</td>
<td>12</td>
<td>9.1E+03</td>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>246</td>
<td>4</td>
<td>1</td>
<td>1.0128171043E+00</td>
<td>1.2E-07</td>
<td>13</td>
<td>1.0E+00</td>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>247</td>
<td>4</td>
<td>1</td>
<td>1.0128171043E+00</td>
<td>5.7E-08</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Optimal solution. Reduced gradient less than tolerance.

The first few lines identify the version of CONOPT you use.

The first iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represents the point that is the result of CONOPT’s pre-processing, and iteration 2 represents the same point after scaling (even if scaling is turned off).

The remaining iterations are characterized by the value of "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 (labeled "Infeasibility") is being minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4 (now labeled "Objective"), is minimized or maximized. Phase 0 iterations are Newton-like iterations. They are very cheap so you should not be concerned if there are many of these Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by a number of inner SLP iterations in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinear and most aspects of the
iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. For small and simple models CONOPT will approximate second order information as a byproduct of the line searches. For larger and more complex models CONOPT will use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These SQP iterations are identified by the number of inner SQP iterations in the "InItr" column.

The column "NSB" for Number of SuperBasics defines the degree of freedom or the dimension of the current search space, and "rgmax" measures the largest reduced gradient among the non-optimal variables. Rgmax should eventually converge towards zero, but convergence is not expected to be monotone. The last two columns labeled "MX" and "OK" gives information about the line search: OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated before an optimal step length was found because it was not possible to find a feasible solution for large step lengths. MX = T means that the line search was terminated by a variable reaching a bound (always combined with OK = T), and MX = F means that the step length was determined by nonlinearities. If OK = T then the step length was optimal; if OK = F then the constraints were too nonlinear to allow CONOPT to make a full optimal step.

5.7.3 GAMS/CONOPT Termination Messages

GAMS/CONOPT may terminate in a number of ways. This section will show most of the termination messages and explain their meaning. It will also show the Model Status returned to GAMS in `<model>.ModelStat`, where `<model>` represents the name of the GAMS model. The Solver Status returned in `<model>.SolveStat` will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation. The first 4 messages are used for optimal solutions and CONOPT will return ModelStat = 2 (Locally Optimal), except as noted below:

** Optimal solution. There are no superbasic variables.

The solution is a locally optimal corner solution. The solution is determined by constraints only, and it is usually very accurate. In some cases CONOPT can determine that the solution is globally optimal and it will return ModelStat = 1 (Optimal).

** Optimal solution. Reduced gradient less than tolerance.

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the tolerance `rtredg` with default value 1.e-7. The value of the objective function is very accurate while the values of the variables are less accurate due to a flat objective function in the interior of the feasible area.

** Optimal solution. The error on the optimal objective function value estimated from the reduced gradient and the estimated Hessian is less than the minimal tolerance on the objective.

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the tolerance `rtredg`. However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. For this to happen the objective must be large or the reduced objective must have large second derivatives so it is advisable to scale the model. See the sections on "Scaling" and "Using the Scale Option in GAMS" for details on how to scale a model.
** Optimal solution. Convergence too slow. The change in objective has been less than xx.xx for xx consecutive iterations.

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance \texttt{rtredg}, but less than \texttt{rtredg} multiplied by the largest Jacobian element divided by 100. The model must have large derivatives so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and ModelStat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that a Sum of Infeasibility objective function is locally minimal, but positive. If the model is convex it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on "Initial Values" for hints on what to do.

** Feasible solution. Convergence too slow. The change in objective has been less than xx.xx for xx consecutive iterations.

** Feasible solution. The tolerances are minimal and there is no change in objective although the reduced gradient is greater than the tolerance.

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all. However, the optimality criteria have not been satisfied. These messages are accompanied by ModelStat = 7 (Feasible Solution) and SolveStat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section NLP and DNLP Models. The problem can also be caused by a poorly scaled model. See section Scaling for hints on model scaling. Finally, it can be caused by stalling as described in section Stalling in Appendix A. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". ModelStat is in this case 6 (Intermediate Infeasible) and SolveStat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well defined local minimum.

** Unbounded solution. A variable has reached 'infinity'.
Largest legal value (Rtmaxv) is xx.xx

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value of \texttt{rtmaxv} (default \texttt{1.e10}) and it returns with ModelStat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable "<var>" mentioned in the first line of the message. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, \texttt{rtmaxv}, as mentioned in the section on options. However, you will pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from "infinity", CONOPT will actually have to make a line search and move to a region with large values of the variables. This may lead to bad scaling and to many different kinds of tolerance and roundoff problems, including problems of determining whether a solution is feasible or not.

The message above exists in a version where "Unbounded" is replaced by "Infeasible" and ModelStat is 5 (Locally Infeasible). You may also see a message like
5.7 CONOPT

**<var>: Free variable becomes too large**

- Free variable becomes too large

**Infeasible solution. A free variable exceeds the allowable range. Current value is 1.02E+10 and current upper bound (Rtmaxv) is 1.00E+10**

These two messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable "<var>" mentioned in the first line of the message. If the model seems correct you should scale it.

**The time limit has been reached.**

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by Option ResLim = xx; or <model>.ResLim = xx; statements, has been reached. CONOPT will return with SolveStat = 3 (Resource Interrupt) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

**The iteration limit has been reached.**

The iteration limit defined in GAMS, either by default (usually 2000000000 iterations) or by Option IterLim = xx; or <model>.IterLim = xx; statements, has been reached. CONOPT will return with SolveStat = 2 (Iteration Interrupt) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

**Domain errors in nonlinear functions.**

- Check bounds on variables.

- The number of function evaluation errors has reached the limit defined in GAMS by Option DomLim = xx; or <model>.DomLim = xx; statements or the default limit of 0 function evaluation errors. CONOPT will return with SolveStat = 5 (Evaluation Error Limit) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution). See more details in section Function Evaluation Errors.

**An initial derivative is too large (larger than Rtmaxj= xx.xx)**

- Scale the variables and/or equations or add bounds.

- **<var> appearing in**

- **<equ>: Initial Jacobian element too large = xx.xx**

and

**A derivative is too large (larger than Rtmaxj= xx.xx).**

- Scale the variables and/or equations or add bounds.

- **<var> appearing in**

- **<equ>: Jacobian element too large = xx.xx**

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will show you where to look. A large derivative means that the function changes very rapidly even after a very small change in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop and you are advised to adjust the model if at all possible.

If the offending derivative is associated with a Log(x) or 1/x term you may try to increase the lower bound on x. If the offending derivative is associated with an Exp(x) term you must decrease the upper bound on x. You may also try to scale the model, either manually or using the variable.Scale and/or equation.Scale option in GAMS as described in section Scaling. There is also in this case a lazy solution: increase the limit on Jacobian elements, rtmaxj; however, you will pay through reduced reliability or longer solution times.

In addition to the messages shown above you may see messages like
** An equation in the pre-triangular part of the model cannot be solved because the critical variable is at a bound.

** An equation in the pre-triangular part of the model cannot be solved because of too small pivot.

or

** An equation is inconsistent with other equations in the pre-triangular part of the model.

These messages containing the word "Pre-triangular" are all related to infeasibilities identified by CONOPT's pre-processing stage and they are explained in detail in section Iteration 1: Preprocessing in Appendix A.

Usually, CONOPT will be able to estimate the amount of memory needed for the model based on statistics provided by GAMS. However, in some cases with unusual models, e.g. very dense models or very large models, the estimate will be too small and you must request more memory yourself using a statement like `<model>.WorkFactor = x.x;` or `<model>.WorkSpace = xx;` in GAMS or by adding `WorkFactor=xx` to the command line call of GAMS. The message you will see is similar to the following:

** FATAL ERROR ** Insufficient memory to continue the optimization.

You must request more memory.
Current CONOPT space = 0.29 Mbytes
Estimated CONOPT space = 0.64 Mbytes
Minimum CONOPT space = 0.33 Mbytes

CONOPT time Total 0.109 seconds
of which: Function evaluations 0.000 = 0.0%
1st derivative evaluations 0.000 = 0.0%

The text after "Insufficient memory to" may be different; it says something about where CONOPT ran out of memory. If the memory problem appears during model setup the message will be accompanied by SolveStat = 13 (System Failure) and ModelStat = 13 (Error No Solution) and CONOPT will not return any values. If the memory problem appears later during the optimization SolveStat will be 11 (Internal Solver Failure) and ModelStat will be either 6 (Intermediate Infeasible) or 7 (Feasible Solution) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

It is recommended that you use the WorkFactor option if you must change the amount of memory. The same number will usually work for a whole family of models. If you prefer to use WorkSpace, the GAMS WorkSpace option corresponds to the amount of memory, measured in Mbytes.

A new termination message has been added from version 3.16C:

** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).

and similar with Infeasible. To prevent non-sensible results and/or infinite loops in special degenerate cases CONOPT has added checks for internal intermediate results being NaN (Not A Number) or very large. If this happens CONOPT will try to change some tolerances and try to continue the optimization. If this attempt fails CONOPT will stop and return the message above. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible." Section Overflow and NaN (Not A Number) in Appendix A has more details on the sources of NaN and the actions that can be taken by the user and by CONOPT.
5.7.4 Function Evaluation Errors

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. Log is not defined for negative arguments, Exp overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable variable bounds. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds.

In some cases bounds are not sufficient, e.g. in the expression Log( Sum(i, x(i) ) ): in some models each individual x should be allowed to become zero, but the Sum should not. In this case you should introduce an intermediate variable and an extra equation, e.g. xSumDef .. xSum =E= sum(i,x(i)); add a lower bound on xSum; and use xSum as the argument to the Log function. See section Simple Expressions for additional comments on this topic.

Whenever a nonlinear function is called outside its domain of definition, GAMS’ function evaluator will intercept the function evaluation error and prevent the system to crash. GAMS will replace the undefined result by some appropriate real number, and it will make sure the error is reported to the modeler as part of the standard solution output in the GAMS listing file. GAMS will also report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Finally, CONOPT will be instructed to stop after DomLim errors.

During Phase 0, 1, and 3 CONOPT will often use large steps as the initial step in a line search and functions will very likely be called with some of the variables at their lower or upper bound. You are therefore likely to get a division-by-zero error if your model contains a division by x and x has a lower bound of zero. And you are likely to get an exponentiation overflow error if your model contains Exp(x) and x has no upper bound. However, CONOPT will usually not get trapped in a point outside the domain of definition for the model. When GAMS’ function evaluator reports that a point is ”bad”, CONOPT will decrease the step length, and it will for most models be able to recover and continue to an optimal solution. It is therefore safe to use a large value for DomLim instead of GAMS default value of 0.

CONOPT may get stuck in some cases, for example because there is no previous point to backtrack to, because ”bad” points are very close to ”reasonable” feasible points, or because the derivatives are not defined in a feasible point. The more common messages are:

** Fatal Error ** Function error in initial point in Phase 0 procedure.

** Fatal Error ** Function error after small step in Phase 0 procedure.

** Fatal Error ** Function error very close to a feasible point.

** Fatal Error ** Function error while reducing tolerances.

** Fatal Error ** Function error in Pre-triangular equations.

** Fatal Error ** Function error after Preprocessing.

** Fatal Error ** Function error in Post-triangular equation.

In the first four cases you must either add better bounds or define better initial values. If the problem is related to a pre- or post-triangular equation as shown by the last three messages then you can turn part of the pre-processing off as described in section Iteration 1: Preprocessing . in Appendix A. However, this may make the model harder to solve, so it is usually better to add bounds and/or initial values.
5.7.5 The CONOPT Options File

CONOPT has been designed to be self-tuning. Most tolerances are dynamic. As an example: The feasibility of a constraint is always judged relative to the dual variable on the constraint and relative to the expected change in objective in the coming iteration. If the dual variable is large then the constraint must be satisfied with a small tolerance, and if the dual variable is small then the tolerance is larger. When the expected change in objective in the first iterations is large then the feasibility tolerances are also large. And when we approach the optimum and the expected change in objective becomes smaller then the feasibility tolerances become smaller.

Because of the self-tuning nature of CONOPT you should in most cases be well off with default tolerances. If you do need to change some tolerances, possibly following the advice in Appendix A, it can be done in the CONOPT Options file. The name of the CONOPT Options file is on most systems "conopt.opt". You must tell the solver that you want to use an options file with the statement <model>.OptFile = 1 in your GAMS source file before the Solve statement or with OptFile = 1 on the command line.

The format of the CONOPT Options file consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e12
lfnsup = 500
```

The value must be written using legal GAMS format, i.e. a real number may contain an optional E exponent, but a number may not contain blanks. The value must have the same type as the option, i.e. real options must be assigned real values, integer options must be assigned integer values, and logical options must be assigned logical values. The logical value representing true are true, t, yes, or 1, and the logical values representing false are false, f, no, or 0.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported. You can still replace the equal sign with := and you can add end of line comments after a # or ! character. Lines starting with * in column 1 are treated as comment lines.

5.7.6 Hints on Good Model Formulation

This section will contain some comments on how to formulate a nonlinear model so it becomes easier to solve with CONOPT. Most of the recommendations will be useful for any nonlinear solver, but not all. We will try to mention when a recommendation is CONOPT specific.

5.7.6.1 Initial Values

Good initial values are important for many reasons. Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution. Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort. The progress of the optimization algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and the initial point in which the initial derivatives are computed is therefore again important. Finally, non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

The initial values used by CONOPT are all coming from GAMS. The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero. Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. An initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable. CONOPT will
warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.

If a variable has a small positive lower bound, for example because it appears as an argument to the Log function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.

You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS. An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS’ scaling option. A better possibility is to select reasonable values for some variables that from the context are known to be important, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

\[
\text{pmDef(it)} \quad \text{pm(it) =e= pwm(it)*er*(1 + tm(it))};
\]

where pm, pwm, and er are variables and tm is a parameter. The following assignment statements use the equation to derive consistent initial values for PM from sensible initial values for pwm and er:

\[
er.l = 1;\quad \text{pwm.l(it)} = 1;
\]

\[
\text{pm.l(it)} = \text{pwm.l(it)}*\text{er.l}*(1 + \text{tm(it)});
\]

With these assignments equation pmDef will be feasible in the initial point, and since CONOPT uses a feasible path method it will remain feasible throughout the optimization (unless the pre-processor destroys it, see section Iteration 1: Preprocessing in Appendix A).

If CONOPT has difficulties finding a feasible solution for your model you should try to use this technique to create an initial point in which as many equations as possible are satisfied. You may also try the optional Crash procedure described in section Preprocessing: The Optional Crash Procedure in Appendix A by adding the line `lstcrs=t` to the CONOPT options file. The crash procedure tries to identify equations with a mixture of uninitialized variables and variables with initial values, and it solves the equations with respect to the uninitialized variables; the effect is similar to the manual procedure shown above.

### 5.7.6.2 Bounds

Bounds have two purposes in nonlinear models. Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems. Algorithmic bounds require a closer look at the functional form of the model. The content of a Log should be greater than say 1.e-3, the content of an Exp should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both Log(x) and 1/x are extremely nonlinear for small arguments. The first and second derivatives of Log(x) at x=1.e-3 are 1.e+3 and -1.e6, respectively, and the first and second derivatives of 1/x at x=1.e-2 are -1.e+4 and 2.e+6, respectively.

If the content of a Log or Exp function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed in the next section.

Note that bounds in some cases can slow the solution process down. Too many bounds may for example introduce degeneracy. If you have constraints of the following type
vub(i) .. x(i) =l= y;

or

ysum .. y =e= sum( i, x(i) );

and x is a Positive Variable then you should in general not declare y a Positive Variable or add a lower bound of zero on y. If y appears in a nonlinear function you may need a strictly positive bound. Otherwise, you should declare y a free variable; CONOPT will then make y basic in the initial point and y will remain basic throughout the optimization. New logic in CONOPT tries to remove this problem by detecting when a harmful bound is redundant so it can be removed, but it is not yet a fool proof procedure.

Section Iteration 1: Preprocessing in Appendix A gives another example of bounds that can be counter productive.

5.7.6.3 Simple Expressions

The following model component

Parameter mu(i);
Variable x(i), s(i), obj;
Equation objDef;
objDef .. obj =e= Exp( Sum( i, Sqr( x(i) - mu(i) ) / s(i) ) );

can be re-written in the slightly longer but simpler form

Parameter mu(i);
Variable x(i), s(i), obj, inTerm;
Equation intDef, objDef;
intDef .. inTerm =e= Sum( i, Sqr( x(i) - mu(i) ) / s(i) );
objDef .. obj =e= Exp( inTerm );

The first formulation has very complex derivatives because Exp is taken of a long expression. The second formulation has much simpler derivatives; Exp is taken of a single variable, and the variables in intDef appear in a sum of simple independent terms.

In general, try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables. Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable. The model will become larger, but the increased size is taken care of by CONOPT's sparse matrix routines, and it is compensated by the reduced complexity. If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

The reduction in complexity can be significant if an intermediate expression is linear. The following model fragment:

Variable x(i), y;
Equation yDef;
yDef .. y =e= 1 / Sum( i, x(i) );
should be written as

```
Variable x(i), xSum, y;
Equation xSumDef, yDef;
xSumDef .. xSum =e= Sum(i, x(i) );
yDef .. y =e= 1 / xSum;
xSum.lo = 1.e-2;
```

for three reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. Second, because the lower bound on the intermediate result will bound the search away from the singularity at xSum = 0. And third, because the matrix of second derivatives for the last model only depend on xSum while it depends on all x in the first model.

The last example shows an added potential saving by expanding functions of linear expressions. A constraint depends in a nonlinear fashion on the accumulated investments, inv, like

```
con(i) .. f( Sum( j$(ord(j) le ord(i)), inv(j) ) ) <= b(i);
```

A new intermediate variable, cap(i), that is equal to the content of the Sum can be defined recursively with the constraints

```
cDef(i) .. cap(i) =e= inv(i) + cap(i-1);
```

and the original constraints become

```
con(i) .. f( cap(i) ) =l= b(i);
```

The reformulated model has n additional variables and n additional linear constraints. In return, the original n complex nonlinear constraints have been changed into n simpler nonlinear constraints. And the number of Jacobian elements, that has a direct influence on much of the computational work both in GAMS and in CONOPT, has been reduced from n*(n+1)/2 nonlinear elements to 3*n-1 linear elements and only n nonlinear element. If f is an invertible increasing function you may even rewrite the last constraint as a simple bound:

```
cap.10(i) = finv(b(i));
```

Some NLP solvers encourage you to move as many nonlinearities as possible into the objective which may make the objective very complex. This is neither recommended nor necessary with CONOPT. A special pre-processing step (discussed in section Iteration 1: Preprocessing in Appendix A) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.
5.7.6.4 Scaling

Nonlinear as well as Linear Programming Algorithms use the derivatives of the objective function and the constraints to determine good search directions, and they use function values to determine if constraints are satisfied or not. The scaling of the variables and constraints, i.e. the units of measurement used for the variables and constraints, determine the relative size of the derivatives and of the function values and thereby also the search path taken by the algorithm.

Assume for example that two goods of equal importance both cost $1 per kg. The first is measured in gram, the second in tons. The coefficients in the cost function will be $1000/g and $0.001/ton, respectively. If cost is measured in $1000 units then the coefficients will be 1 and 1.e-6, and the smaller may be ignored by the algorithm since it is comparable to some of the zero tolerances.

CONOPT assumes implicitly that the model to be solved is well scaled. In this context well scaled means:

- Basic and superbasic solution values are expected to be around 1, e.g. from 0.01 to 100. Nonbasic variables will be at a bound, and the bound values should not be larger than say 100.
- Dual variables (or marginals) on active constraints are expected to be around 1, e.g. from 0.01 to 100. Dual variables on non-binding constraints will of course be zero.
- Derivatives (or Jacobian elements) are expected to be around 1, e.g. from 0.01 to 100.

Variables become well scaled if they are measured in appropriate units. In most cases you should select the unit of measurement for the variables so their expected value is around unity. Of course there will always be some variation. Assume x(i) is the production at location i. In most cases you should select the same unit of measurement for all components of x, for example a value around the average capacity.

Equations become well scaled if the individual terms are measured in appropriate units. After you have selected units for the variables you should select the unit of measurement for the equations so the expected values of the individual terms are around one. If you follow these rules, material balance equations will usually have coefficients of plus and minus one.

Derivatives will usually be well scaled whenever the variables and equations are well scaled. To see if the derivatives are well scaled, run your model with a positive Option LimRow and look for very large or very small coefficients in the equation listing in the GAMS output file.

CONOPT computes a measure of the scaling of the Jacobian, both in the initial and in the final point, and if it seems large it will be printed. The message looks like:

** WARNING ** The variance of the derivatives in the initial point is large (= 4.1 ). A better initial point, a better scaling, or better bounds on the variables will probably help the optimization.

The variance is computed as Sqrt(Sum(Log(Abs(jac(i)))**2)/nz) where jac(i) represents the nz nonzero derivatives (Jacobian elements) in the model. A variance of 4.1 corresponds to an average value of Log(jac)**2 of 4.1**2, which means that Jacobian values outside the range Exp(-4.1)=0.017 to Exp(+4.1)=60.4 are about as common at values inside. This range is for most models acceptable, while a variance of 5, corresponding to about half the derivatives outside the range Exp(-5)=0.0067 to Exp(+5)=148, can be dangerous.
5.7.6.4.1 Scaling of Intermediate Variables Many models have a set of variables with a real economic or physical interpretation plus a set of intermediate or helping variables that are used to simplify the model. We have seen some of these in section Simple Expressions on Simple Expressions. It is usually rather easy to select good scaling units for the real variables since we know their order of magnitude from economic or physical considerations. However, the intermediate variables and their defining equations should preferably also be well scaled, even if they do not have an immediate interpretation. Consider the following model fragment where \( x, y, \) and \( z \) are variables and \( y \) is the intermediate variable:

```
Set p / p0*p4 /
Parameter a(p) / p0 211, p1 103, p2 42, p3 31, p4 6 /
yDef .. y =e= Sum(p, a(p)*Power(x,Ord(p)-1));
zDef .. z =e= Log(y);
```

\( x \) lies in the interval 1 to 10 which means that \( y \) will be between 211 and 96441 and \( Z \) will be between 5.35 and 11.47. Both \( x \) and \( z \) are reasonably scaled while \( y \) and the terms and derivatives in \( yDef \) are about a factor 1.e4 too large. Scaling \( y \) by 1.e4 and renaming it \( ys \) gives the following scaled version of the model fragment:

```
yDefs1 .. ys =e= Sum(p, a(p)*Power(x,Ord(p)-1))*1.e-4;
zDefs1 .. z =e= Log(ys*1.e4);
```

The \( z \) equation can also be written as

```
zDefs2 .. z =e= Log(ys) + Log(1.e4);
```

Note that the scale factor 1.e-4 in the \( yDefs1 \) equation has been placed on the right hand side. The mathematically equivalent equation

```
yDefs2 .. ys*1.e4 =e= Sum(p, a(p)*Power(x,Ord(p)-1));
```

will give a well scaled \( ys \), but the right hand side terms of the equation and their derivatives have not changed from the original equation \( yDef \) and they are still far too large.

5.7.6.4.2 Using the Scale Option in GAMS The rules for good scaling mentioned above are exclusively based on algorithmic needs. GAMS has been developed to improve the effectiveness of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as "natural" as possible. The units of measurement is one part of this natural notation, and there is unfortunately often a conflict between what the modeler thinks is a good unit and what constitutes a well scaled model.

To facilitate the translation between a natural model and a well scaled model GAMS has introduced the concept of a scale factor, both for variables and equations. The notation and the definitions are quite simple. First of all, scaling is by default turned off. To turn it on, enter the statement `<model>.ScaleOpt = 1;` in your GAMS program somewhere after the `Model` statement and before the `Solve` statement. `<model>` is the name of the model to be solved. If you want to turn scaling off again, enter the statement `<model>.ScaleOpt = 0;` somewhere before the next `Solve`.

The scale factor of a variable or an equation is referenced with the suffix ".Scale", i.e. the scale factor of variable \( x(i) \) is referenced as \( x.Scale(i) \). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined in assignment statements with \( x.Scale(i) \) on the left hand side, and scale factors, both from variables and equations, can be used on the right hand side, for example to define other scale factors. The default scale factor is always 1, and a scale factor must be positive; GAMS will generate an execution time error if the scale factor is less than 1.e-20.

The mathematical definition of scale factors is as follows: The scale factor on a variable, \( V^x \) is used to related the variable as seen by the modeler, \( V^m \), to the variable as seen by the algorithm, \( V^a \), as follows:
\[ V^m = V^a \times V^s \]

This means, that if the variable scale, \( V^s \), is chosen to represent the order of magnitude of the modeler's variable, \( V^m \), then the variable seen by the algorithm, \( V^a \), will be around 1. The scale factor on an equation, \( G^s \), is used to related the equation as seen by the modeler, \( G^m \), to the equation as seen by the algorithm, \( G^a \), as follows:

\[ G^m = G^a \times G^s \]

This means, that if the equation scale, \( G^s \), is chosen to represent the order of magnitude of the individual terms in the modeler's version of the equation, \( G^m \), then the terms seen by the algorithm, \( G^a \), will be around 1.

The derivatives in the scaled model seen by the algorithm, i.e. \( dG^a/dV^a \), are related to the derivatives in the modeler's model, \( dG^m/dV^m \), through the formula:

\[ dG^a/dV^a = dG^m/dV^m \times V^s/G^s \]

i.e. the modelers derivative is multiplied by the scale factor of the variable and divided by the scale factor of the equation. Note, that the derivative is unchanged if \( V^s = G^s \). Therefore, if you have a GAMS equation like

\[ G .. V =e= \text{expression}; \]

and you select \( G^s = V^s \) then the derivative of \( V \) will remain 1. If we apply these rules to the example above with an intermediate variable we can get the following automatic scale calculation, based on an "average" reference value for \( x \):

```gams
Scalar xRef; xRef = 6;
y.Scale = Sum(p, a(p)*Power(xRef,Ord(p)-1));
yDef.Scale = y.Scale;
```

or we could scale \( y \) using values at the end of the \( x \) interval and add safeguards as follows:

```gams
y.Scale = Max( Abs(Sum(p, a(p)*Power(x.Lo,Ord(p)-1))),
                 Abs(Sum(p, a(p)*Power(x.Up,Ord(p)-1)));
```

Lower and upper bounds on variables are automatically scaled in the same way as the variable itself. Integer and binary variables cannot be scaled.

GAMS' scaling is in most respects hidden for the modeler. The solution values reported back from a solution algorithm, both primal and dual, are always reported in the user's notation. The algorithm's versions of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if Option LimRow and/or LimCol are positive, and in debugging output from the solution algorithm, generated with Option SysOut = On. In addition, the numbers in the algorithms iteration log will represent the scaled model: the infeasibilities and reduced gradients will correspond to the scaled model, and if the objective variable is scaled, the value of the objective function will be the scaled value.

A final warning about scaling of multidimensional variables is appropriate. Assume variable \( x(i,j,k) \) only appears in the model when the parameter \( ijk(i,j,k) \) is nonzero, and assume that \( \text{Card}(i) = \text{Card}(j) = \text{Card}(k) = 100 \) while \( \text{Card}(ijk) \) is much smaller than \( 100^3 = 1.e6 \). Then you should only scale the variables that appear in the model, i.e.

```gams
x.Scale(i,j,k)$ijk(i,j,k) = expression;
```

The statement

```gams
x.Scale(i,j,k) = expression;
```

will generate records for \( x \) in the GAMS database for all combinations of \( i, j, \) and \( k \) for which the expression is different from 1, i.e. up to \( 1.e6 \) records, and apart from spending a lot of time you will very likely run out of memory. Note that this warning also applies to non-default lower and upper bounds.
5.7.7 NLP and DNLP Models

GAMS has two classes of nonlinear model, NLP and DNLP. NLP models are defined as models in which all functions that appear with endogenous arguments, i.e. arguments that depend on model variables, are smooth with smooth derivatives. DNLP models can in addition use functions that are smooth but have discontinuous derivatives. The usual arithmetic operators (+, -, ∗, /, and ∗∗) can appear on both model classes.

The functions that can be used with endogenous arguments in a DNLP model and not in an NLP model are Abs, Min, and Max and as a consequence the indexed operators SMin and SMax.

Note that the offending functions can be applied to expressions that only involve constants such as parameters, var.l, and equ.m. Fixed variables are in principle constants, but GAMS makes its tests based on the functional form of a model, ignoring numerical parameter values and numerical bound values, and terms involving fixed variables can therefore not be used with Abs, Min, or Max in an NLP model.

The NLP solvers used by GAMS can also be applied to DNLP models. However, it is important to know that the NLP solvers attempt to solve the DNLP model as if it was an NLP model. The solver uses the derivatives of the constraints with respect to the variables to guide the search, and it ignores the fact that some of the derivatives may change discontinuously. There are at the moment no GAMS solvers designed specifically for DNLP models and no solvers that take into account the discontinuous nature of the derivatives in a DNLP model.

5.7.7.1 DNLP Models: What Can Go Wrong?

Solvers for NLP Models are all based on making marginal improvements to some initial solution until some optimality conditions ensure no direction with marginal improvements exist. A point with no marginally improving direction is called a Local Optimum.

The theory about marginal improvements is based on the assumption that the derivatives of the constraints with respect to the variables are a good approximations to the marginal changes in some neighborhood around the current point.

Consider the simple NLP model, Min Sqr(x), where x is a free variable. The marginal change in the objective is the derivative of Sqr(x) with respect to x, which is 2x. At x = 0, the marginal change in all directions is zero and x = 0 is therefore a Local Optimum.

Next consider the simple DNLP model, Min Abs(x), where x again is a free variable. The marginal change in the objective is still the derivative, which is +1 if x > 0 and -1 if x < 0. When x = 0, the derivative depends on whether we are going to increase or decrease x. Internally in the DNLP solver, we cannot be sure whether the derivative at 0 will be -1 or +1; it can depend on rounding tolerances. An NLP solver will start in some initial point, say x = 1, and look at the derivative, here +1. Since the derivative is positive, x is reduced to reduce the objective. After some iterations, x will be zero or very close to zero. The derivative will be +1 or -1, so the solver will try to change x. however, even small changes will not lead to a better objective function. The point x = 0 does not look like a Local Optimum, even though it is a Local Optimum. The result is that the NLP solver will muddle around for some time and then stop with a message saying something like: "The solution cannot be improved, but it does not appear to be optimal."

In this first case we got the optimal solution so we can just ignore the message. However, consider the following simple two-dimensional DNLP model: Min Abs(x1+x2) + 5∗Abs(x1-x2) with x1 and x2 free variables. Start the optimization from x1 = x2 = 1. Small increases in x1 will increase both terms and small decreases in x1 (by dx) will decrease the first term by dx but it will increase the second term by 5∗dx. Any change in x1 only is therefore bad, and it is easy to see that any change in x2 only also is bad. An NLP solver may therefore be stuck in the point x1 = x2 = 1, even though it is not a local solution: the direction (dx1, dx2) = (-1,-1) will lead to the optimum in x1 = x2 = 0. However, the NLP solver cannot distinguish what happens with this model from what happened in the previous model; the message will be of the same type: "The solution cannot be improved, but it does not appear to be optimal."
5.7.7.2 Reformulation from DNLP to NLP

The only reliable way to solve a DNLP model is to reformulate it as an equivalent smooth NLP model. Unfortunately, it may not always be possible. In this section we will give some examples of reformulations.

The standard reformulation approach for the ABS function is to introduce positive and negative deviations as extra variables: The term \( z = \text{Abs}(f(x)) \) is replaced by \( z = f_{\text{Plus}} + f_{\text{Minus}} \), \( f_{\text{Plus}} \) and \( f_{\text{Minus}} \) are declared as positive variables and they are defined with the identity: \( f(x) = f_{\text{Plus}} - f_{\text{Minus}} \). The discontinuous derivative from the Abs function has disappeared and the part of the model shown here is smooth. The discontinuity has been converted into lower bounds on the new variables, but bounds are handled routinely by any NLP solver. The feasible space is larger than before; \( f(x) = 5 \) can be obtained both with \( f_{\text{Plus}} = 5, f_{\text{Minus}} = 0 \), and \( z = 5 \), and with \( f_{\text{Plus}} = 1000, f_{\text{Minus}} = 995 \), and \( z = 1995 \). Provided the objective function has some term that tries to minimize \( z \), either \( f_{\text{Plus}} \) or \( f_{\text{Minus}} \) will become zero and \( z \) will end with its proper value.

You may think that adding the smooth constraint \( f_{\text{Plus}} \cdot f_{\text{Minus}} = 0 \) would ensure that either \( f_{\text{Plus}} \) or \( f_{\text{Minus}} \) is zero. However, this type of so-called complementarity constraint is "bad" in any NLP model. The feasible space consists of the two half lines: \( (f_{\text{Plus}} = 0 \) and \( f_{\text{Minus}} \geq 0 \) ) and \( (f_{\text{Plus}} \geq 0 \) and \( f_{\text{Minus}} = 0 \). Unfortunately, the marginal change methods used by most NLP solvers cannot move from one half line to the other, and the solution is stuck at the half line it happens to reach first.

There is also a standard reformulation approach for the Max function. The equation \( z = \max(f(x),g(y)) \) is replace by the two inequalities, \( z = g \cdot f(x) \) and \( z = g \cdot g(y) \). Provided the objective function has some term that tries to minimize \( z \), one of the constraints will become binding as equality and \( z \) will indeed be the maximum of the two terms.

The reformulation for the Min function is similar. The equation \( z = \min(f(x),g(y)) \) is replaced by the two inequalities, \( z = l \cdot f(x) \) and \( z = l \cdot g(y) \). Provided the objective function has some term that tries to maximize \( z \), one of the constraints will become binding as equality and \( z \) is indeed the minimum of the two terms.

Max and Min can have more than two arguments and the extension should be obvious.

The non-smooth indexed operators, SMax and SMin can be handled using a similar technique: for example, \( z = \text{SMax}(i, f(x,i)) \) is replaced by the indexed inequality: \( \text{inEq}(i) \cdot z = l \cdot f(x,i) \);

The reformulations that are suggested here all enlarge the feasible space. They require the objective function to move the final solution to the intersection of this larger space with the original feasible space. Unfortunately, the objective function is not always so helpful. If it is not, you may try using one of the smooth approximations described next. However, you should realize, that if the objective function cannot help the "good" approximations described here, then your overall model is definitely non-convex and it is likely to have multiple local optima.

5.7.7.3 Smooth Approximations

Smooth approximations to the non-smooth functions ABS, MAX, and MIN are approximations that have function values close to the original functions, but have smooth derivatives.

A smooth GAMS approximation for Abs(f(x)) is

\[
\text{Sqrt}( \text{Sqr}(f(x)) + \text{Sqr}(\delta) )
\]
where delta is a small scalar. The value of delta can be used to control the accuracy of the approximation and the curvature around \( f(x) = 0 \). The approximation error is largest when \( f(x) \) is zero, in which case the error is delta. The error is reduced to approximately \( \sqrt{\delta^2 + \delta^2} / 2 \) for \( f(x) = 1 \). The second derivative is \( 1/\delta \) at \( f(x) = 0 \) (excluding terms related to the second derivative of \( f(x) \)). A delta value between 1.e-3 and 1.e-4 should in most cases be appropriate. It is possible to use a larger value in an initial optimization, reduce it and solve the model again. You should note, that if you reduce delta below 1.e-4 then large second order terms might lead to slow convergence or even prevent convergence.

The approximation shown above has its largest error when \( f(x) = 0 \) and smaller errors when \( f(x) \) is far from zero. If it is important to get accurate values of Abs exactly when \( f(x) = 0 \), then you may use the alternative approximation

\[
\sqrt{\Delta (f(x)^2 + \delta^2)} - \delta
\]

instead. The only difference is the constant term. The error is zero when \( f(x) \) is zero and the error grows to \( -\delta \) when \( f(x) \) is far from zero.

Some theoretical work uses the Huber, \( H(\cdot) \), function as an approximation for Abs. The Huber function is defined as

\[
H(x) = \begin{cases} 
  x & \text{for } x > \delta, \\
  -x & \text{for } x < -\delta \text{ and} \\
  \frac{\sqrt{x^2}}{2\delta} + \frac{\delta}{2} & \text{for } -\delta < x < \delta.
\end{cases}
\]

Although the Huber function has some nice properties, it is for example accurate when \( \text{Abs}(x) > \delta \), it is not so useful for GAMS work because it is defined with different formula for the three pieces.

A smooth GAMS approximation for \( \text{Max}(f(x),g(y)) \) is

\[
( f(x) + g(y) + \sqrt{\Delta (f(x)-g(y)^2 + \delta^2)} )/2
\]

where delta again is a small scalar. The approximation error is delta/2 when \( f(x) = g(y) \) and decreases with the difference between the two terms. As before, you may subtract a constant term to shift the approximation error from the area \( f(x) = g(y) \) to areas where the difference is large. The resulting approximation becomes

\[
( f(x) + g(y) + \sqrt{\Delta (f(x)-g(y)^2 + \delta^2)} - \delta )/2
\]

Similar smooth GAMS approximations for \( \text{Min}(f(x),g(y)) \) are

\[
( f(x) + g(y) - \sqrt{\Delta (f(x)-g(y)^2 + \delta^2)} )/2
\]

and

\[
( f(x) + g(y) - \sqrt{\Delta (f(x)-g(y)^2 + \delta^2)} + \delta )/2
\]

Appropriate delta values are the same as for the Abs approximation: in the range from 1.e-2 to 1.e-4.

It appears that there are no simple symmetric extensions for Max and Min of three or more arguments or for indexed SMax and SMin.
5.7.7.4 Are DNLP Models Always Non-smooth?

A DNLP model is defined as a model that has an equation with an Abs, Max, or Min function with endogenous arguments. The non-smooth properties of DNLP models are derived from the non-smooth properties of these functions through the use of the chain rule. However, composite expressions involving Abs, Max, or Min can in some cases have smooth derivatives and the model can therefore in some cases be smooth.

One example of a smooth expression involving an Abs function is common in water systems modeling. The pressure loss over a pipe, \( dH \), is proportional to the flow, \( Q \), to some power, \( P \). \( P \) is usually around +2. The sign of the loss depend on the direction of the flow so \( dH \) is positive if \( Q \) is positive and negative if \( Q \) is negative. Although GAMS has a Sign function, it cannot be used in a model because of its discontinuous nature. Instead, the pressure loss can be modeled with the equation

\[
dH = \text{const} \ast Q \ast \text{Abs}(Q)^{(P-1)}
\]

where the sign of the Q-term takes care of the sign of \( dH \), and the Abs function guaranties that the real power \( ^{(P-1)} \) is applied to a non-negative number. Although the expression involves the Abs function, the derivatives are smooth as long as \( P \) is greater than 1. The derivative with respect to \( Q \) is \( \text{const} \ast (P-1) \ast \text{Abs}(Q)^{(P-1)} \) for \( Q > 0 \) and \( -\text{const} \ast (P-1) \ast \text{Abs}(Q)^{(P-1)} \) for \( Q < 0 \). The limit for \( Q \) going to zero from both right and left is 0, so the derivative is smooth in the critical point \( Q = 0 \) and the overall model is therefore smooth.

Another example of a smooth expression is the following terribly looking Sigmoid expression:

\[
\text{Sigmoid}(x) = \frac{\text{Exp}(\text{Min}(x,0))}{1+\text{Exp}(-\text{Abs}(x))}
\]

The standard definition of the sigmoid function is

\[
\text{Sigmoid}(x) = \frac{\text{Exp}(x)}{1+\text{Exp}(x)}
\]

This definition is well behaved for negative and small positive \( x \), but it not well behaved for large positive \( x \) since \( \text{Exp} \) overflows. The alternative definition:

\[
\text{Sigmoid}(x) = \frac{1}{1+\text{Exp}(-x)}
\]

is well behaved for positive and slightly negative \( x \), but it overflows for very negative \( x \). Ideally, we would like to select the first expression when \( x \) is negative and the second when \( x \) is positive, i.e.

\[
\text{Sigmoid}(x) = \frac{\text{Exp}(x)/(1+\text{Exp}(x))}{(x < 0)} + \frac{(1/(1+\text{Exp}(-x)))}{(x > 0)}
\]

but a \$\$ -control that depends on an endogenous variable is illegal. The first expression above solves this problem. When \( x \) is negative, the nominator becomes \( \text{Exp}(x) \) and the denominator becomes \( 1+\text{Exp}(x) \). And when \( x \) is positive, the nominator becomes \( \text{Exp}(0) = 1 \) and the denominator becomes \( 1+\text{Exp}(-x) \). Since the two expressions are mathematically identical, the combined expression is of course smooth, and the \( \text{Exp} \) function is never evaluated for a positive argument.

Unfortunately, GAMS cannot recognize this and similar special cases so you must always solve models with endogenous Abs, Max, or Min as DNLP models, even in the cases where the model is smooth.
5.7 CONOPT

5.7.7.5 Are NLP Models Always Smooth?

NLP models are defined as models in which all operators and functions are smooth. The derivatives of composite functions, that can be derived using the chain rule, will therefore in general be smooth. However, it is not always the case. The following simple composite function is not smooth: \( y = \text{Sqrt}(\text{Sqr}(x)) \). The composite function is equivalent to \( y = \text{Abs}(x) \), one of the non-smooth DNLP functions.

What went wrong? The chain rule for computing derivatives of a composite function assumes that all intermediate expressions are well defined. However, the derivative of Sqrt grows without bound when the argument approaches zero, violating the assumption.

There are not many cases that can lead to non-smooth composite functions, and they are all related to the case above: The real power, \( x^{**y} \), for \( 0 < y < 1 \) and \( x \) approaching zero. The Sqrt function is a special case since it is equivalent to \( x^{**y} \) for \( y = 0.5 \).

If you have expressions involving a real power with an exponent between 0 and 1 or a Sqrt, you should in most cases add bounds to your variables to ensure that the derivative or any intermediate terms used in their calculation become undefined. In the example above, Sqrt( Sqr(x) ), a bound on \( x \) is not possible since \( x \) should be allowed to be both positive and negative. Instead, changing the expression to Sqrt( Sqr(x) + Sqr(delta) ) may lead to an appropriate smooth formulation.

Again, GAMS cannot recognize the potential danger in an expression involving a real power, and the presence of a real power operator is not considered enough to flag a model as a DNLP model. During the solution process, the NLP solver will compute constraint values and derivatives in various points within the bounds defined by the modeler. If these calculations result in undefined intermediate or final values, a function evaluation error is reported, an error counter is incremented, and the point is flagged as a bad point. The following action will then depend on the solver. The solver may try to continue, but only if the modeler has allowed it with an `Option DomLim = xxx;`. The problem of detecting discontinuities is changed from a structural test at the GAMS model generation stage to a dynamic test during the solution process.

You may have a perfectly nice model in which intermediate terms become undefined. The composite function Sqrt( Power(x,3) ) is mathematically well defined around \( x = 0 \), but the computation will involve the derivative of Sqrt at zero, that is undefined. It is the modeler's responsibility to write expressions in a way that avoids undefined intermediate terms in the function and derivatives computations. In this case, you may either add a small strictly positive lower bound on \( x \) or rewrite the function as \( x^{**1.5} \).

5.7.8 APPENDIX A: Algorithmic Information

The objective of this Appendix is to give technically oriented users some understanding of what CONOPT is doing so they can get more information out of the iteration log. This information can be used to prevent or circumvent algorithmic difficulties or to make informed guesses about which options to experiment with to improve CONOPT's performance on particular model classes.

5.7.8.1 Overview of GAMS/CONOPT

GAMS/CONOPT is a GRG-based algorithm specifically designed for large nonlinear programming problems expressed in the following form

\[
\begin{align*}
\text{min or max} & \quad f(x) \\
\text{subject to} & \quad g(x) = b \\
\text{lo} & \quad < x < \text{up}
\end{align*}
\]
where \( x \) is the vector of optimization variables, \( \text{lo} \) and \( \text{up} \) are vectors of lower and upper bounds, some of which may be minus or plus infinity, \( b \) is a vector of right hand sides, and \( f \) and \( g \) are differentiable nonlinear functions that define the model. \( n \) will in the following denote the number of variables and \( m \) the number of equations. (2) will be referred to as the (general) constraints and (3) as the bounds.

The relationship between the mathematical model in (1)-(3) above and the GAMS model is simple: The inequalities defined in GAMS with \( =l= \) or \( =g= \) are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all GAMS equalities to ensure that the Jacobian matrix, i.e. the matrix of derivatives of the functions \( g \) with respect to the variables \( x \), has full row rank. All these slacks are together with the normal GAMS variables included in \( x \). \( \text{lo} \) represent the lower bounds defined in GAMS, either implicitly with the Positive Variable declaration, or explicitly with the Var.Lo notation, as well as any bounds on the slacks. Similarly, \( \text{up} \) represent upper bounds defined in GAMS, e.g. with the Var.Up notation, as well as any bounds on the slacks. \( g \) represent the non-constant terms of the GAMS equations themselves; non-constant terms appearing on the right hand side are moved to the left hand side and constant terms on the left hand side are moved to the right. The objective function \( f \) is simply the GAMS variable to be minimized or maximized.

Additional comments on assumptions and design criteria can be found in the Introduction to the main text.

5.7.8.2 The CONOPT Algorithm

The algorithm used in GAMS/CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details on the algorithm can be found in Drud (1985 and 1992). Here we will just give a short verbal description of the major steps in a generic GRG algorithm. The later sections in this Appendix will discuss some of the enhancements in CONOPT that make it possible to solve large models.

The key steps in any GRG algorithm are:

1. Initialize and Find a feasible solution.
2. Compute the Jacobian of the constraints, \( J \).
3. Select a set of \( n \) basic variables, \( x_b \), such that \( B \), the sub-matrix of basic column from \( J \), is nonsingular. Factorize \( B \). The remaining variables, \( x_n \), are called nonbasic.
4. Solve \( B^T u = df/dx_b \) for the multipliers \( u \).
5. Compute the reduced gradient, \( r = df/dx - J^T u \). \( r \) will by definition be zero for the basic variables.
6. If \( r \) projected on the bounds is small, then stop. The current point is close to optimal.
7. Select the set of superbasic variables, \( x_s \), as a subset of the nonbasic variables that profitably can be changed, and find a search direction, \( d_s \), for the superbasic variables based on \( r_s \) and possibly on some second order information.
8. Perform a line search along the direction \( d \). For each step, \( x_s \) is changed in the direction \( d_s \) and \( x_b \) is subsequently adjusted to satisfy \( g(x_b, x_s) = b \) in a pseudo-Newton process using the factored \( B \) from step 3.
9. Go to 2.
The individual steps are of course much more detailed in a practical implementation like CONOPT. Step 1 consists of several pre-processing steps as well as a special Phase 0 and a scaling procedure as described in the following sections. The optimizing steps are specialized in several versions according to the whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the matrices $J$ and $B$ can be avoided or done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 can be improved by observing that the optimal step as in LP almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized: the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this (section Finding a Feasible Solution: Phase 1 and 2 and Linear and Nonlinear Mode: Phase 1 to 4). If the model is "very" linear an improved search direction (step 7) can be computed using specialized inner LP-like iterations (section Linear Mode: The SLP Procedure) and a steepest edge procedure can be useful for certain models that needs very many iterations (section Linear Mode: The Steepest Edge Procedure). If the model is "very" nonlinear and has many degrees of freedom an improved search direction (step 7) can be computed using specialized inner SQP-like iterations based on exact second derivatives for the model (section Nonlinear Mode: The SQP Procedure).

The remaining two sections give some short guidelines for selecting non-default options (section How to Select Non-default Options), and discuss miscellaneous topics (section Miscellaneous Topics) such as CONOPT's facilities for strictly triangular models (section Triangular Models) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear System (section Constrained Nonlinear System or Square Systems of Equations), as well as numerical difficulties due to loss of feasibility (section Loss of Feasibility) and slow or no progress due to stalling (section Stalling).

5.7.8.3 Iteration 0: The Initial Point

The first few "iterations" in the iteration log (See section Iteration Output), in the main text for an example) are special initialization iterations, but they have been counted as real iterations to allow the user to interrupt at various stages during initialization. Iteration 0 corresponds to the input point exactly as it was received from GAMS. The sum of infeasibilities in the column labeled "Infeasibility" includes all residuals, also from the objective constraint where "z =e= expression" will give rise to the term $\text{Abs}(z - \text{expression})$ that may be nonzero if $z$ has not been initialized. You may stop CONOPT after iteration 0 with Option IterLim = 0; in GAMS. The solution returned to GAMS will contain the input point and the values of the constraints in this point. The marginals of both variables and equations have not yet been computed and they will be returned as EPS.

This possibility can be used for debugging when you have a reference point that should be feasible, but is infeasible for unknown reasons. Initialize all variables to their reference values, also all intermediate variables, and call CONOPT with IterLim = 0. Then compute and display the following measures of infeasibility for each block of constraints, represented by the generic name eq:

\[
\begin{align*}
\text{=} & \text{ constraints: Round}(\text{Abs}(\text{eq.L} - \text{eq.Lo}),3) \\
\text{\leq} & \text{ constraints: Round}(\text{Min}(0,\text{eq.L} - \text{eq.Up}),3) \\
\text{\geq} & \text{ constraints: Round}(\text{Min}(0,\text{eq.Lo} - \text{eq.L}),3)
\end{align*}
\]

The Round function rounds to 3 decimal places so GAMS will only display the infeasibilities that are larger than 5.e-4.

Similar information can be derived from inspection of the equation listing generated by GAMS with Option LimRow = nn; but although the method of going via CONOPT requires a little more work during implementation it can be convenient in many cases, for example for large models and for automated model checking.
5.7.8.4 Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraint-variable pairs that can be solved a priori (so-called pre-triangular equations and variables) are solved and the corresponding variables are assigned their final values. Constraints that always can be made feasible because they contain a free variable with a constant coefficient (so-called post-triangular equation-variable pairs) are excluded from the search for a feasible solution and from the Infeasibility measure in the iteration log. Implicitly, equations and variables are ordered as shown in Figure 1.

5.7.8.4.1 Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Figure 1. They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, GAMS/CONOPT looks for equations with only one non-fixed variable. If such an equation exists, GAMS/CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section Triangular Models.

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```gams
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;
```
Equation \( e2 \) is first solved with respect to \( x2 \) (result \( 3/5 = 0.6 \)). It is easy to solve the equation since \( x2 \) appears linearly, and the result will be unique. \( x2 \) is then fixed and the equation is removed. Equation \( e1 \) is now a candidate since \( x1 \) is the only remaining non-fixed variable in the equation. Here \( x1 \) appears nonlinear and the value of \( x1 \) is found using an iterative scheme based on Newton's method. The iterations are started from the value provided by the modeler or from the default initial value. In this case \( x1 \) is started from the default initial value, i.e. the lower bound of 0.1, and the result after some iterations is \( x1 = 2.718 = \text{Exp}(1) \).

During the recursive solution process it may not be possible to solve one of the equations. If the lower bound on \( x1 \) in the model above is changed to 3.0 you will get the following output:

** An equation in the pre-triangular part of the model cannot be solved because the critical variable is at a bound.

- Residual= 9.86122887E-02
- Tolerance (RTNWTR)= 6.34931126E-07

\( e1: \) Infeasibility in pre-triangular part of model.
\( x1: \) Infeasibility in pre-triangular part of model.

The solution order of the critical equations and variables is:

\( e2 \) is solved with respect to \( x2 \). Solution value = 6.0000000000E-01

\( e1 \) could not be solved with respect to \( x1 \). Final solution value = 3.0000000000E+00
\( e1 \) remains infeasible with residual = 9.8612288668E-02

The problem is as indicated that the variable to be solved for is at a bound, and the value suggested by Newton's method is on the infeasible side of the bound. The critical variable is \( x1 \) and the critical equation is \( e1 \), i.e. \( x1 \) tries to exceed its bound when CONOPT solves equation \( e1 \) with respect to \( x1 \). To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility: In this case equation \( e2 \) was first solved with respect to variable \( x2 \), then equation \( e1 \) was attempted to be solved with respect to \( x1 \) at which stage the problem appeared. To make the analysis easier CONOPT will always report the minimal set of equations and variables that caused the infeasibility.

Another type of infeasibility is shown by the following model:

Variable \( x1, x2, x3, \) obj;
Equation \( e1, e2, e3; \)
\( e1 .. \text{Sqr}(x1) + x2 =e= 1.6; \)
\( e2 .. 5 * x2 =e= 3; \)
\( e3 .. \text{obj} =e= \text{Sqr}(x1) + 2 * \text{Sqr}(x2) + 3 * \text{Sqr}(x3); \)
Model demo / All /; Solve demo using NLP Minimizing obj;

where \( \log(x1) \) has been replaced by \( \text{Sqr}(x1) \) and the lower bound on \( x1 \) has been removed. This model gives the message:

** An equation in the pre-triangular part of the model cannot be solved because of too small pivot. Adding a bound or initial value may help.

- Residual= 4.0000000
Tolerance (RTNWTR)= 6.34931126E-07

e1: Infeasibility in pre-triangular part of model.
x1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and variables is:

e2 is solved with respect to
x2. Solution value = 6.0000000000E-01

e1 could not be solved with respect to
x1. Final solution value = 0.0000000000E+00

After equation e2 has been solved with respect to x2, equation e1 that contains the term Sqr(x) should be solved with respect to x1. The initial value of x1 is the default value zero. The derivative of e1 with respect to x1 is therefore zero, and it is not possible for CONOPT to determine whether to increase or decrease x1. If x1 is given a nonzero initial value the model will solve. If x1 is given a positive initial value the equation will give x1 = 1, and if x1 is given a negative initial value the equation will give x1 = -1. The last type of infeasibility that can be detected during the solution of the pre-triangular or recursive equations is shown by the following example

Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
e4 .. x1 + x2 =e= 3.318;
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;

that is derived from the first model by the addition of equation e4. This model produces the following output

** An equation is inconsistent with other equations in the pre-triangular part of the model.
Residual= 2.81828458E-04
Tolerance (RTNWTR)= 6.34931126E-07

The pre-triangular feasibility tolerance may be relaxed with a line:

    SET RTNWTR X.XX

in the CONOPT control program.

e4: Inconsistency in pre-triangular part of model.

The solution order of the critical equations and variables is:

e2 is solved with respect to
x2. Solution value = 6.0000000000E-01

e1 is solved with respect to
x1. Solution value = 2.7182818285E+00

All variables in equation e4 are now fixed
and the equation is infeasible. Residual = 2.8182845830E-04

First e2 is solved with respect to x2, then e1 is solved with respect to x1 as indicated by the last part of
the output. At this point all variables that appear in equation e4, namely x1 and x2, are fixed, but the
equation is not feasible. e4 is therefore inconsistent with e1 and e2 as indicated by the first part of the
output. In this case the inconsistency is fairly small, 2.8E-04, so it could be a tolerance problem. CONOPT
will always report the tolerance that was used, rtnewtr - the triangular Newton tolerance, and if the
infeasibility is small it will also tell how the tolerance can be relaxed. Section The CONOPT Options File
gives further details on how to change tolerances, and a complete list of options is given in Appendix B.

You can turn the identification and solution of pre-triangular variables and equations off by adding the
line lspret = f in the CONOPT control program. This can be useful in some special cases where the
point defined by the pre-triangular equations gives a function evaluation error in the remaining equations.
The following example shows this:

Variable x1, x2, x3, x4, obj;
Equation e1, e2, e3, e4;
e1 .. Log(1+x1) + x2 =e= 0;
e2 .. 5 * x2 =e= -3;
e3 .. obj =e= 1*Sqr(x1) + 2*Sqrt(0.01 + x2 - x4) + 3*Sqr(x3);
e4 .. x4 =l= x2;
Model fer / All /; Solve fer Minimizing obj using NLP;

All the nonlinear functions are defined in the initial point in which all variables have their default value
of zero. The pre-processor will compute x2 = -0.6 from e2 and x1 = 0.822 from e1. When CONOPT
continues and attempts to evaluate e3, the argument to the Sqrt function is negative when these new
triangular values are used together with the initial x4 = 0, and CONOPT cannot backtrack to some safe
point since the function evaluation error appears the first time e3 is evaluated. When the pre-triangular
preprocessor is turned off, x2 and x4 are changed at the same time and the argument to the Sqrt function
remains positive throughout the computations. Note, that although the purpose of the e4 inequality is to
guarantee that the argument of the Sqrt function is positive in all points, and although e4 is satisfied
in the initial point, it is not satisfied after the pre-triangular constraints have been solved. Only simple
bounds are strictly enforced at all times. Also note that if the option lspret = f is used then feasible
linear constraints will in fact remain feasible.

An alternative (and preferable) way of avoiding the function evaluation error is to define an intermediate
variable equal to 0.01+x2-x4 and add a lower bound of 0.01 on this variable. The inequality e4 could then
be removed and the overall model would have the same number of constraints.

5.7.8.4.2 Preprocessing: Post-triangular Variables and Constraints  Consider the following
fragment of a larger GAMS model:

Variable util(t) Utility in period t
totutil Total Utility;
Equation utilDef(T) Definition of Utility
tutilDef Definition of Total Utility;
utilDef(T).. util(t) =e= nonlinear function of other variables;
tutilDef .. totutil =e= Sum( t, util(t) / (1+r)**Ord(t) );
Model demo / All /; Solve demo Maximizing totutil using NLP;
The part of the model shown here is easy to read and from a modeling point of view it should be considered well written. However, it could be more difficult to solve than a model in which variable \( \text{util}(t) \) was substituted out because all the \( \text{utilDef} \) equations are nonlinear constraints that the algorithms must ensure are satisfied.

To make well written models like this easy to solve CONOPT will move as many nonlinearities as possible from the constraints to the objective function. This automatically changes the model from the form that is preferable for the modeler to the form that is preferable for the algorithm. In this process CONOPT looks for free variables that only appear in one equation outside the objective function. If such a variable exists and it appears linearly in the equation, like \( \text{util}(t) \) appears with coefficient 1 in equation \( \text{utilDef}(t) \), then the equation can always be solved with respect to the variable. This means that the variable logically can be substituted out of the model and the equation can be removed. The result is a model that has one variable and one equation less, and a more complex objective function. As variables and equations are substituted out, new candidates for elimination may emerge, so CONOPT repeats the process until no more candidates exist.

This so-called post-triangular preprocessing step will often move several nonlinear constraints into the objective function where they are much easier to handle, and the effective size of the model will decrease. In some cases the result can even be a model without any general constraints. The name post-triangular is derived from the way the equations and variables appear in the permuted Jacobian in Figure 1. The post-triangular equations and variables are the ones on the lower right hand corner labeled B and II, respectively.

In the example above, the \( \text{util} \) variables will be substituted out of the model together with the nonlinear \( \text{utilDef} \) equations provided the \( \text{util} \) variables are free and do not appear elsewhere in the model. The resulting model will have fewer nonlinear constraints, but more nonlinear terms in the objective function.

Although you may know that the nonlinear functions on the right hand side of \( \text{utilDef} \) always will produce positive \( \text{util} \) values, you should in general not declare \( \text{util} \) to be a Positive Variable. If you do, GAMS/CONOPT may not be able to eliminate \( \text{util}(t) \), and the model will be harder to solve. It is of course unfortunate that a redundant bound changes the solution behavior, and to reduce this problem CONOPT will try to estimate the range of nonlinear expressions using interval arithmetic. If the computed range of the right hand side of the \( \text{utilDef} \) constraint is within the bounds of \( \text{util} \), then these bounds cannot be binding and \( \text{util} \) is a so-called implied free variable that can be eliminated.

The following model fragment from a least squares model shows another case where the preprocessing step in GAMS/CONOPT is useful:

```gams
Variable residual(case) Residuals
ssq Sum of Squared Residuals;
Equation eqEst(case) Equation to be estimated
ssqDef Definition of objective;
  eqEst(case).. residual(case) =e= expression in other variables;
SSQDEF .. ssq =e= Sum( case, Sqr( residual(case) ) );
Model lsqLarge / All /; Solve lsqLarge using NLP Minimizing ssq;
```

GAMS/CONOPT will substitute the residual variables out of the model using the \( \text{eqEst} \) equations. The model solved by GAMS/CONOPT is therefore mathematically equivalent to the following GAMS model

```gams
Variable ssq Sum of Squared Residuals;
Equation ssqd Definition of objective;
  ssqd .. ssq =e= Sum( case, Sqr(expression in other variables));
Model lsqSmall / All /;
Solve lsqSmall using NLP Minimizing ssq;
```
However, if the "expression in other variables" is a little complicated, e.g. if it depends on several variables, then the first model, lsqLarge, will be much faster to generate with GAMS because its derivatives in equation qEst and ssqDef are much simpler than the derivatives in the combined ssqd equation in the second model, lsqSmall. The larger model will therefore be faster to generate, and it will also be faster to solve because the computation of both first and second derivatives will be faster.

Note that the comments about what are good model formulations are dependent on the preprocessing capabilities in GAMS/CONOPT. Other algorithms may prefer models like lsqSmall over lsqLarge. Also note that the variables and equations that are substituted out are still indirectly part of the model. GAMS/CONOPT evaluates the equations and computes values for the variables each time the value of the objective function is needed, and their values are available in the GAMS solution.

It is not necessary to have a coefficient of 1 for the variable to be substituted out in the post-triangular phase. However, a non-zero coefficient cannot be smaller than the absolute pivot tolerance used by CONOPT, rtpiva.

The number of pre- and post-triangular equations and variables is printed in the log file between iteration 0 and 1 as shown in the iteration log in section Iteration Output of the main text. The sum of infeasibilities will usually decrease from iteration 0 to 1 because fewer constraints usually will be infeasible. However, it may increase as shown by the following example:

```
Positive Variable x, y, z;
Equation e1, e2;
e1.. x =e= 1;
e2.. 10*x - y + z =e= 0;
```

started from the default values x.L = 0, y.L = 0, and z.L = 0. The initial sum of infeasibilities is 1 (from e1 only). During pre-processing x is selected as a pre-triangular variable in equation e1 and it is assigned its final value 1 so e1 becomes feasible. After this change the sum of infeasibilities increases to 10 (from e2 only).

You may stop CONOPT after iteration 1 with Option IterLim = 1; in GAMS. The solution returned to GAMS will contain the pre-processed values for the variables that can be assigned values from the pre-triangular equations, the computed values for the variables used to solve the post-triangular equations, and the input values for all other variables. The pre- and post-triangular constraints will be feasible, and the remaining constraints will have values that correspond to this point. The marginals of both variables and equations have not been computed yet and will be returned as EPS.

The crash procedure described in the following sub-section is an optional part of iteration 1.

### 5.7.8.4.3 Preprocessing: The Optional Crash Procedure

In the initial point given to CONOPT the variables are usually split into a group with initial value provided by the modeler (in the following called the assigned variables) and a group of variables for which no initial value has been provided (in the following called the default variables). The objective of the optional crash procedure is to find a point in which as many of the constraints as possible are feasible, primarily by assigning values to the default variables and by keeping the assigned variables at their initial values. The implicit assumption in this procedure is that if the modeler has assigned an initial value to a variable then this value is "better" than a default initial value.

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a singleton row appears, or until all variables have been fixed. The variables to be fixed at their initial value are selected using a heuristic that both tries to create many row singletons
and tries to select variables with "good values". Since the values of many variables will come to depend in the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.

Figure 2 shows a reordered version of Figure 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g. intermediate variables. The number of crash-triangular variables is shown on the iteration output between iteration 0 and 1, but only if the crash procedure is turned on.

The result of the crash procedure is an updated initial point in which usually a large number of equations will be feasible, namely all equations labeled A, B, and C in Figure 2. There is, as already shown with the small example in Section Preprocessing: Post-triangular Variables and Constraints above, no guarantee that the sum of infeasibilities will be reduced, but it is often the case, and the point will often provide a good starting point for the following procedures that finds an initial feasible solution.

The crash procedure is activated by adding the line \texttt{lstcrs=t} in the options file. The default value of \texttt{lstcrs} (\texttt{lstcrs} = Logical Switch for Triangular Crash) is \texttt{f} or \texttt{false}, i.e. the crash procedure is not normally used. Before turning the crash procedure on you must turn the definitional equations (see next sub-section) off.

5.7.8.4.4 Preprocessing: Definitional Equations From version 3.16 CONOPT has introduced the concept of Definitional Equations. In section Simple Expressions it was recommended to introduce intermediate variables to simplify complex expressions. If the intermediate variables are free or if the bounds defined by the modeler cannot be binding then we call the constraints that define the intermediate variables "Definitional Equations" and the intermediate variables "Defined Variables". Some models have a large number of definitional equations, and CONOPT tries to recognize them and take advantage of the special structure. Defined variables that only appear in or feed into the objective are recognized as post-triangular variables as discussed in section Preprocessing: Post-triangular Variables and Constraints but defined variables can also be used in the simultaneous constraints. The picture is similar to fig 2. with the C-rows representing the definitional constraints and the III-variables the defined variables. The main differences between crash-triangular variables and defined variable are that (1) defined variables are free or have non-binding bounds and the definitional equations can therefore always be made feasible, (2) defined variables are cheaper to recognize, (3) since they have a natural interpretation, using them is probably more numerical stable, and (4) there are most likely fewer defined variables than crash-triangular variables.

The number of definitional equations is printed in the log file between iteration 0 and 1 if CONOPT finds any. The definitional equations are used to give the defined variables new values, so it is no longer so important the give intermediate variables initial values. In the process the sum of infeasibilities may grow but CONOPT consider it more important to keep these constraints feasible. The defined variables are also made basic and they will most likely stay basic throughout the solution process.

There are three new options introduced to controls the detection of definitional equations: \texttt{lsusdf}, \texttt{lsuqdf}, and \texttt{lfusdf}. They are described in Appendix B. By default CONOPT will only look for unique definitional constraints, but the options allow the user to experiment with a more aggressive strategy.

From version 3.16F there is an additional option, \texttt{lmusdf}, used to control how the definitional equations are used during the optimization. The option is summarized in Appendix B. By default, definitional equations are only used to define initial values and an initial basis and their special properties are ignored during the optimization. If there are many definitional equations and they are fairly nonlinear then it can sometimes be beneficial to force the definitional variables to remain basic and to use the definitional equations to compute values for the defined variables in all intermediate trial points. This behavior is turned on with \texttt{lmusdf = 1}. The point where \texttt{lmusdf = 1} start to pay off depends on the model and the degree of nonlinearity, but a guess is that the number of definitional equations should exceed half the number of equations remaining after the pre- and post-triangular equations have been removed before it is worth while.
5.7.8.4.5 Preprocessing and Function evaluation errors  Function evaluation errors may occur during pre-processing is an equation that is a candidate for becoming pre-triangular. Since GAMS requires all functions to be defined in the starting point this can only happen if a previously solved pre-triangular variable has received a new value that makes the function undefined. An example is:

```plaintext
positive variable x1, x2
equation e1, e2;
e1 .. x1 =E= 10;
e2 .. x2 =E= log( 5-x1 );
```

The pre-triangular variable x1 is changed to 10, a value that is inconsistent with equation e2. You can see the changes that CONOPT has made during pre-processing with the option `prprec = true` and this should help identify why the model does not solve.

After identifying and solving the pre-triangular part of the model CONOPT looks for implied bounds, and if an initial variable violates such an implied bound, CONOPT will move it inside the bound. This change can give rise to a function evaluation error in one of the remaining constraints. An example is:

```plaintext
positive variable x1, x2, x3, x4;
equation e1, e2;
e1.. x1 + x2 =L= 10;
e2.. x4 =E= log(x1+x3-20);
x1.l = 100; x3.up = 100;
```

From constraint e1 we can derive that both x1 and x2 must have (implied) upper bounds of 10. Since the initial value of x1 is above this implied bound it is changed by the pre-processor. After the pre-processor finishes, e2 will give a function evaluation error. Again, option `prprec = true` will give output that describes the changes in the variables which should help identify the problem.

5.7.8.5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default is to turn scaling on. The Infeasibility column shows the scaled sum of infeasibilities. You may again stop CONOPT after iteration 2 with `Option IterLim = 2` in GAMS, but the solution that is reported in GAMS will have been scaled back again so there will be no change from iteration 1 to iteration 2.

The following description of the automatic scaling procedure is included for completeness. Experiments have so far given mixed results with some advantage for scaling, and scaling is therefore by default turned on, corresponding to the CONOPT option `lsscal = t`. Users are recommended to be cautious with the automatic scaling procedure. If scaling is a problem, try to use manual scaling or scaling in GAMS (see section Scaling in the main text) based on an understanding of the model.

The scaling procedure multiplies all variables in group III and all constraints in group C (see Figure 1) by scale factors computed as follows:

1. CONOPT computes the largest term for each constraint, i. This is defined as the maximum of the constant right hand side, the slack (if any), and `Abs(jac(i,j)*x(j))` where `jac(i,j)` is the derivative and `x(j)` is the variable.
2. The constraint scale factor is defined as the largest term in the constraint, projected on the interval \([\text{rtmins}, \text{rtmaxs}]\). The constraint is then divided by the constraint scale factor. Ignoring the projection, the result is a model in which the largest term in each constraint is exactly 1. The purpose of the projection is to prevent extreme scaling. The default value of \(\text{rtmins}\) is 1 which implies that we do not scale the constraints up. Constraints with only small terms remain unchanged. The default value of \(\text{rtmaxs}\) is \(2^{*30}\) or around \(1.07e9\) so terms much larger than \(1.e9\) are only partially scaled down and will still remain large.

3. The terms \(\text{Abs}(\text{jac}(i,j)\times x(j))\) after constraint scaling measure the importance of each variable in the particular constraint. The variable scale is selected so the largest importance of the variable over all constraints is 1. This gives a very simple variable scale factor, namely the absolute value of the variable. The variable is then divided by the variable scale factor. To avoid extreme scaling we again project on the interval \([\text{rtmins}, \text{rtmaxs}]\). Variables less than \(\text{rtmins}\) (default 1) are therefore not scaled up and variables over \(\text{rtmaxs}\) (default \(2^{*30} = 1.07e9\)) are only partially scaled down.

You should note that CONOPT by default scales large numbers down, but it does not scale small numbers up. You should therefore try to avoid having variables or terms in expressions that are small but significant. If this is not possible, allow CONOPT to scale up by giving the option \(\text{rtmins}\) a value less than 1.

All scale factors are rounded down to a power of 2 to preserve precision in the internal computations. To avoid difficulties with rapidly varying variables and derivatives CONOPT recomputes the scale factors at regular intervals (see \(\text{lfscal}\)).

The options that control scaling, \(\text{lsscal}, \text{lfscal}, \text{rtmins},\) and \(\text{rtmaxs}\), are all described in Appendix B.

5.7.8.6 Finding a Feasible Solution: Phase 0

The GRG algorithm used by CONOPT is a feasible path algorithm. This means that once it has found a feasible point it tries to remain feasible and follow a path of improving feasible points until it reaches a local optimum. CONOPT starts with the point provided by GAMS. This point will always satisfy the bounds (3): GAMS will simply move a variable that is outside its bounds to the nearer bound before it is presented to the solver. If the general constraints (2) also are feasible then CONOPT will work with feasible solutions throughout the optimization. However, the initial point may not satisfy the general constraints (2). If this is not the case, GAMS/CONOPT must first find an initial feasible point. This first step can be just as hard as finding an optimum for some models. For some models feasibility is the only problem.

GAMS/CONOPT has two methods for finding an initial feasible point. The first method is not very reliable but it is fast when it works; the second method is reliable but slower. The fast method is called Phase 0 and it is described in this section. It is used first. The reliable method, called Phase 1 and 2, will be used if Phase 0 terminates without a feasible solution.

Phase 0 is based on the observation that Newton's method for solving a set of equations usually is very fast, but it may not always converge. Newton's method in its pure form is defined for a model with the same number of variables as equations, and no bounds on the variables. With our type of model there are usually too many variables, i.e. too many degrees of freedom, and there are bounds. To get around the problem of too many variables, GAMS/CONOPT selects a subset with exactly \(m\) "basic" variables to be changed. The rest of the variables will remain fixed at their current values, that are not necessarily at bounds. To accommodate the bounds, GAMS/CONOPT will try to select variables that are away from their bounds as basic, subject to the requirement that the Basis matrix, consisting of the corresponding columns in the Jacobian, must have full rank and be well conditioned.

The Newton equations are solved to yield a vector of proposed changes for the basic variables. If the full proposed step can be applied we can hope for the fast convergence of Newton's method. However, several things may go wrong:
1. The infeasibilities, measured by the 1-norm of g (i.e. the sum of the absolute infeasibilities, excluding the pre- and post-triangular equations), may not decrease as expected due to nonlinearities.

2. The maximum step length may have to be reduced if a basic variable otherwise would exceed one of its bounds.

In case 1. GAMS/CONOPT tries various heuristics to find a more appropriate set of basic variables. If this does not work, some "difficult" equations, i.e. equations with large infeasibilities and significant nonlinearities, are temporarily removed from the model, and Newton's method is applied to the remaining set of "easy" equations.

In case 2. GAMS/CONOPT will remove the basic variable that first reaches one of its bounds from the basis and replace it by one of the nonbasic variables. Newton's method is then applied to the new set of basic variables. The logic is very close to that of the dual simplex method. In cases where some of the basic variables are exactly at a bound GAMS/CONOPT uses an anti degeneracy procedure based on Ryan and Osborne (1988) to prevent cycling.

Phase 0 will end when all equations except possibly some "difficult" equations are feasible within some small tolerance. If there are no difficult equations, GAMS/CONOPT has found a feasible solution and it will proceed with Phase 3 and 4. Otherwise, Phase 1 and 2 is used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: Iter, Phase, Ninf, Infeasibility, Step, MX, and OK. The number in the Ninf column counts the number of "difficult" infeasible equations, and the number in the Infeasibility column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. There are three possible combinations of values in the MX and OK columns: combination (1) has F in the MX column and T in the OK column and it will always be combined with 1.0 in the Step column: this is an ideal Newton step. The infeasibilities in the easy equations should be reduced quickly, but the difficult equations may dominate the number in the Infeasibility column so you may not observe it. However, a few of these iterations is usually enough to terminate Phase 0. Combination (2) has T in the MX column indicating that a basic variable has reached its bound and is removed from the basis as in case 2. above. This will always be combined with T in the OK column. The Step column will show a step length less than the ideal Newton step of 1.0. Combination (3) has F in both the MX and OK column. It is the bad case and will always be combined with a step of 0.0: this is an iteration where nonlinearities are dominating and one of the heuristics from case 1. must be used.

The success of the Phase 0 procedure is based on being able to choose a good basis that will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so GAMS/CONOPT has some variables away from their bounds to select from. This topic was discussed in more detail in section Initial Values.

The start and the iterations of Phase 0 can, in addition to the crash option described in section Finding a Feasible Solution: Phase 0 be controlled with the three options lslack, lsmxbs, and lmmxsf described in Appendix B.

5.7.8.7 Finding a Feasible Solution: Phase 1 and 2

Most of the equations will be feasible when phase 0 stops. To remove the remaining infeasibilities CONOPT uses a procedure similar to the phase 1 procedure used in Linear Programming: artificial variables are added to the infeasible equations (the equations with Large Residuals), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variable are already part of the model as slack variables; their bounds are simply relaxed temporarily.

This infeasibility minimization problem is similar to the overall optimization problem: minimize an objective function subject to equality constraints and bounds on the variables. The feasibility problem is therefore solved with the ordinary GRG optimization procedure. As the artificial variables gradually become zero, i.e. as the infeasible equations become feasible, they are taken out of the auxiliary objective function. The number of infeasibilities (shown in the Ninf column of the log file) and the sum of infeasibilities (in the Infeasibility column) will therefore both decrease monotonically.

The iteration output will label these iterations as phase 1 and/or phase 2. The distinction between phases 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phases 3 and 4, which are described in the next sections.
5.7.8.8 Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in The CONOPT Algorithm above. The factorization in step 3 is performed using an efficient sparse LU factorization similar to the one described by Suhl and Suhl (1990). The matrix operations in step 4 and 5 are also performed sparse.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model is locally. When the model appears to be fairly linear in the area in which the optimization is performed, i.e., when the function and constraint values are close to their linear approximation for the steps that are taken, then CONOPT takes advantage of the linearity. The derivatives (the Jacobian) are not computed in every iteration, the basis factorization is updated using cheap LP techniques as described by Reid (1982), the search direction is determined without use of second order information, i.e. similar to a steepest descent algorithm, and the initial step length is estimated as the step length where the first variable reaches a bound; very often, this is the only step length that has to be evaluated. These cheap almost linear iterations are referred to as Linear Mode and they are labeled Phase 1 when the model is infeasible and objective is the sum of infeasibilities and Phase 3 when the model is feasible and the real objective function is optimized.

When the constraints and/or the objective appear to be more nonlinear CONOPT will still follow step 2 to 9 of the GRG algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed in each iteration since the nonlinearities imply that the derivatives change. On the other hand, the set of basic variables will often be the same and CONOPT will take advantage of this during the factorization of the basis. In step 7 CONOPT uses the BFGS algorithm to estimate second order information and determine search directions. And in step 8 it will often be necessary to perform more than one step in the line search. These nonlinear iterations are labeled Phase 2 in the output if the solution is still infeasible, and Phase 4 if it is feasible. The iterations in phase 2 and 4 are in general more expensive than the iteration in phase 1 and 3.

Some models will remain in phase 1 (linear mode) until a feasible solution is found and then continue in phase 3 until the optimum is found, even if the model is truly nonlinear. However, most nonlinear models will have some iterations in phase 2 and/or 4 (nonlinear mode). Phase 2 and 4 indicates that the model has significant nonlinear terms around the current point: the objective or the constraints deviate significantly from a linear model for the steps that are taken. To improve the rate of convergence CONOPT tries to estimate second order information in the form of an estimated reduced Hessian using the BFGS formula.

Each iteration is, in addition to the step length shown in column "Step", characterized by two logicals: MX and OK. MX = T means that the step was maximal, i.e. it was determined by a variable reaching a bound. This is the expected value in Phase 1 and 3. MX = F means that no variable reached a bound and the optimal step length will in general be determined by nonlinearities. OK = T means that the line search was well-behaved and an optimal step length was found; OK = F means that the line search was ill-behaved, which means that CONOPT would like to take a larger step, but the feasibility restoring Newton process used during the line search did not converge for large step lengths. Iterations marked with OK = F (and therefore also with MX = F) will usually be expensive, while iterations marked with MX = T and OK = T will be cheap.

5.7.8.9 Linear Mode: The SLP Procedure

When the model continues to appear linear CONOPT will often take many small steps, each determined by a new variable reaching a bound. Although the line searches are fast in linear mode, each require one or more evaluations of the nonlinear constraints, and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraint evaluations CONOPT may replace the steepest descend direction in step 7 of the GRG algorithm with a sequential linear programming (SLP) technique to find a search direction that anticipates the bounds on all variables and therefore gives a larger expected change in objective in each line search. The search direction and the last basis from the SLP procedure are used in an ordinary GRG-type line search in which the solution is made feasible at each step. The
SLP procedure is only used to generate good directions; the usual feasibility preserving steps in CONOPT are maintained, so CONOPT is still a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SLP-mode are identified by numbers in the column labeled InItr in the iteration log. The number in the InItr column is the number of non-degenerate SLP iterations. This number is adjusted dynamically according to the success of the previous iterations and the perceived linearity of the model.

The SLP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model. This is indicated with MX = T and OK = T.
- The objective is nonlinear along the search direction and the optimal step is less than one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SLP procedure or not, based on progress information. You may turn it off completely with the line lseslp = f in the CONOPT options file (usually conopt.opt). The default value of lseslp (l seslp = Logical Switch Enabling SLP mode) is t or true, i.e. the SLP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define lseslp, but it can be useful if CONOPT repeatedly turns SLP on and off, i.e. if you see a mixture of lines in the iteration log with and without numbers in the InItr column.

5.7.8.10 Linear Mode: The Steepest Edge Procedure

When optimizing in linear mode (Phase 1 or 3) CONOPT will by default use a steepest descend algorithm to determine the search direction. CONOPT allows you to use a Steepest Edge Algorithm as an alternative. The idea, borrowed from Linear Programming, is to scale the nonbasic variables according to the Euclidean norm of the “updated column” in a standard LP tableau, the so-called edge length. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables which has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length, and the large change in basic variables is more likely to give rise to larger nonlinear terms.

The steepest edge algorithm has been very successful for linear programs, and our initial experience has also shown that it will give fewer iterations for most nonlinear models. However, the cost of maintaining the edge lengths can be more expensive in the nonlinear case and it depends on the model whether steepest edge results in faster overall solution times or not. CONOPT uses the updating methods for the edge lengths from LP, but it must re-initialize the edge lengths more frequently, e.g. when an inversion fails, which happens more frequently in nonlinear models than in linear models, especially in models with many product terms, e.g. blending models, where the rank of the Jacobian can change from point to point.

Steepest edge is turned on with the line l sanrm = t, in the CONOPT options file (usually conopt.opt). The default value of l sanrm (l san rm = Logical Switch for A- Norm) is f or false, i.e. the steepest edge procedure is turned off.

The steepest edge procedure is mainly useful during linear mode iterations. However, it has some influence in phase 2 and 4 also: The estimated reduced Hessian in the BFGS method is initialized to a diagonal matrix with elements on the diagonal computed from the edge lengths, instead of the usual scaled unit matrix.
5.7.8.11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities CONOPT needs second order information. Some second order information can be derived from the line search and is used in the first iterations in Phase 2 or 4. Depending on progress, CONOPT may switch to a Sequential Quadratic Programming (SQP) procedure that works on a sub-model with linear constraints and a quadratic objective function. The constraints are a linearization of the nonlinear constraints, and the objective function is derived from the Hessian of the Lagrangian function. CONOPT will inside the SQP procedure use exact second order information computed by GAMS. The result of the SQP procedure is a search direction and a basis and CONOPT will afterwards use the same line search procedure and feasibility preserving steps as after the SLP procedure.

CONOPT remains a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SQP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The number in the InItr column is the number of non-degenerate SQP iterations. The effort spend inside the SQP procedure is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

The SQP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model of the constraints. This is indicated with MX = T and OK = T.
- The objective is much more nonlinear along the search direction than expected and the optimal step is not one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SQP procedure or not, based on progress information. You may turn it off completely with the line \texttt{lsesqp = f} in the CONOPT options file (usually \texttt{conopt.opt}). The default value of \texttt{lsesqp} (\texttt{lsesqp = Logical Switch Enabling SQP mode}) is \texttt{t} or \texttt{true}, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define \texttt{lsesqp}, but it can be used for experimentation.

In connection with 1st and 2nd derivatives the listing file (*.lst) will have a few extra lines. The first looks as follows:

```
The model has 537 variables and 457 constraints
with 1597 Jacobian elements, 380 of which are nonlinear.
The Hessian of the Lagrangian has 152 elements on the diagonal,
228 elements below the diagonal, and 304 nonlinear variables.
```

The first two lines repeat information given in the GAMS model statistics and the last two lines describe second order information. CONOPT uses the matrix of second derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian). The Hessian is symmetric and the statistics show that it has 152 elements on the diagonal and 228 below for a total of 380 elements in this case. This compares favorably to the number of elements in the matrix of first derivatives (the Jacobian).

For some models you may see the following message instead (before the usual CONOPT banner):

```
** Warning ** Memory Limit for Hessians exceeded.
You can use the Conopt option "rvhess"
```
The creation of the matrix of second derivatives has been interrupted because the matrix became too dense. A dense matrix of second derivatives will be slow to compute and it will need a lot of memory. In addition, it is likely that a dense Hessian will make some of the computations inside the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

GAMS/CONOPT can use second derivatives even if the Hessian is not available. A special version of the function evaluation routine can compute the Hessian multiplied by a vector (the so-called directional second derivative) without computing the Hessian itself. This routine is used when the Hessian is not available. The directional second derivative approach will require one directional second derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

If you get the "Memory Limit for Hessians exceeded" message you may consider rewriting some equation. Look for nonlinear functions applied to long linear or separable expressions such as \( \log(\text{sum}(i,x(i))) \); as discussed in Section Simple Expressions . An expression like this will create a dense Hessian with \( \text{card}(i) \) rows and columns. You should consider introducing an intermediate variable that is equal to the long linear or separable expression and then apply the nonlinear function to this single variable. You may also experiment with allocating more memory for the dense Hessian and use it despite the higher cost; it may reduce the number of iterations. This can be done by adding the option \( \text{rvhess} = \text{XX} \) to the CONOPT options file. rvhess is a memory factor with default value 10 so you need a larger value. The value 0.0 is special; it means do not impose a memory limit on the Hessian.

The time spend on the many types of function and derivative evaluations are reported in the listing file in a section like this:

```
CONOPT time Total 0.734 seconds
  of which: Function evaluations 0.031 = 4.3%
            1st Derivative evaluations 0.020 = 2.7%
            2nd Derivative evaluations 0.113 = 15.4%
            Directional 2nd Derivative 0.016 = 2.1%
```

The function evaluations are computations of the nonlinear terms in the model, and 1st Derivatives evaluations are computations of the Jacobian of the model. 2nd Derivative evaluations are computations of the Hessian of the Lagrangian, and Directional 2nd derivative evaluations are computations of the Hessian multiplied by a vector, computed without computing the Hessian itself. The lines for 2nd derivatives will only be present if CONOPT has used this type of 2nd derivative.

If your model is not likely to benefit from 2nd derivative information or if you know you will run out of memory anyway you can save a small setup cost by telling GAMS/CONOPT not to generate it using option \( \text{dobess} = \text{f} \).

5.7.8.12 How to Select Non-default Options

The non-default options have an influence on different phases of the optimization and you must therefore first observe whether most of the time is spend in Phase 0, Phase 1 and 3, or in Phase 2 and 4.

**Phase 0:** The quality of Phase 0 depends on the number of iterations and on the number and sum of infeasibilities after Phase 0. The iterations in Phase 0 are much faster than the other iterations, but the overall time spend in Phase 0 may still be rather large. If this is the case, or if the infeasibilities after Phase 0 are large you may try to use the triangular crash options:

```
lstcrs = t
```

Observe if the initial sum of infeasibility after iteration 1 has been reduced, and if the number of phase 0 iterations and the number of infeasibilities at the start of phase 1 have been reduced. If lstcrs reduces the initial sum of infeasibilities but the number of iterations still is large you may try:
lslack = t

CONOPT will after the preprocessor immediately add artificial variables to all infeasible constraints so Phase 0 will be eliminated, but the sum and number of infeasibilities at the start of Phase 1 will be larger. You are in reality trading Phase 0 iterations for Phase 1 iterations.

You may also try the experimental bending line search with

lmmxsf = 1

The line search in Phase 0 will with this option be different and the infeasibilities may be reduced faster than with the default lmmxsf = 0. It is likely to be better if the number of iterations with both MX = F and OK~=-~F is large. This option may be combined with lstcrs = t. Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

**Phase 1 and 3:** The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, lsanrm = t, but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but work in progress tries to push this limit upwards. Try it when the number of iterations is very large, or when many iterations are poorly behaved identified with OK = F in the iteration log. The default SLP mode is usually an advantage, but it is too expensive for a few models. If you observe frequent changes between SLP mode and non-SLP mode, or if many line searches in the SLP iterations are ill-behaved with OK = F, then it may be better to turn SLP off with lssslp = f.

**Phase 2 and 4:** There are currently not many options available if most of the time is spend in Phase 2 and Phase 4. If the change in objective during the last iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, rtredg.

5.7.8.13 Miscellaneous Topics

5.7.8.13.1 Triangular Models A triangular model is one in which the non-fixed variables and the equations can be sorted such that the first equation only depends on the first variable, the second equation only depends on the first two variables, and the p-th equation only depends on the first p variables. Provided there are no difficulties with bounds or small pivots, triangular models can be solved one equation at a time using the method describe in section Preprocessing: Pre-triangular Variables and Constraints and the solution process will be very fast and reliable.

Triangular models can in many cases be useful for finding a good initial feasible solution: Fix a subset of the variables so the remaining model is known to be triangular and solve this triangular simulation model. Then reset the bounds on the fixed variables to their original values and solve the original model. The first solve will be very fast and if the fixed variables have been fixed at good values then the solution will also be good. The second solve will start from the good feasible solution generated by the first solve and it will usually optimize much more quickly than from a poor start.

The modeler can instruct CONOPT that a model is supposed to be triangular with the option lstria = t. CONOPT will then use a special version of the preprocessing routine (see section Preprocessing: Pre-triangular Variables and Constraints) that solves the model very efficiently. If the model is solved successfully then CONOPT terminates with the message:

** Feasible solution to a recursive model.**
and the Model Status will be 2, Locally Optimal, or 1, Optimal, depending on whether there were any nonlinear pivots or not. All marginals on both variables and equations are returned as 0 (zero) or EPS.

Two SOLVEs with different option files can be arranged by writing the option files as they are needed from within the GAMS program with PUT statements followed by a PutClose. You can also have two different option files, e.g., conopt.opt and conopt.op2, and select the second with the GAMS statement <model>.OptFile = 2;

The triangular facility handles a number of error situations:

1. Non-triangular models: CONOPT will ensure that the model is indeed triangular. If it is not, CONOPT will return model status 5, Locally Infeasible, plus some information that allows the modeler to identify the mistake. The necessary information is related to the order of the variables and equations and number of occurrences of variables and equations, and since GAMS does no have a natural place for this type of information CONOPT returns it in the marginals of the equations and variables. The solution order for the triangular equations and variables that have been solved successfully are defined with positive numbers in the marginals of the equations and variables. For the remaining non-triangular variables and equations CONOPT shows the number of places they appear as negative numbers, i.e. a negative marginal for an equation shows how many of the non-triangular variables that appear in this equation. You must fix one or more variables until at least one of the non-triangular equation only has one non-fixed variable left.

2. Infeasibilities due to bounds: If some of the triangular equations cannot be solved with respect to their variable because the variable will exceed the bounds, then CONOPT will flag the equation as infeasible, keep the variable at the bound, and continue the triangular solve. The solution to the triangular model will therefore satisfy all bounds and almost all equations. The termination message will be

** Infeasible solution. xx artificial(s) have been introduced into the recursive equations.

and the model status will be 5, Locally Infeasible.

The modeler may in this case add explicit artificial variables with high costs to the infeasible constraints and the resulting point will be an initial feasible point to the overall optimization model. You will often from the mathematics of the model know that only some of the constraints can be infeasible, so you will only need to check whether to add artificials in these equations. Assume that a block of equations matbal(m,t) could become infeasible. Then the artificials that may be needed in this equation can be modeled and identified automatically with the following GAMS constructs:

```
Set aposart(m,t) Add a positive artificial in Matbal
anegart(m,t) Add a negative artificial in Matbal;
aposart(m,t) = No; anegart(m,t) = No;

Positive Variable
vposart(m,t) Positive artificial variable in Matbal
vnegart(m,t) Negative artificial variable in Matbal;

matbal(m,t).. Left hand side =e= right hand side
  + vposart(m,t)$aposart(m,t) - vnegart(m,t)$anegart(m,t);

objDef.. obj =e= other_terms +
  weight * Sum((m,t), vposart(m,t)$aposart(m,t)
  +vnegart(m,t)$anegart(m,t) );

Solve triangular model ...
aposart(m,t)$$(matbal.l(m,t) > matbal.Up(m,t)) = Yes;
anegart(m,t)$$(matbal.l(m,t) < matbal.Lo(m,t)) = Yes;

Solve final model ...
```
3. Small pivots: The triangular facility requires the solution of each equation to be locally unique which also means that the pivots used to solve each equation must be nonzero. The model segment

\[
\begin{align*}
e_1 & \quad x_1 = e = 0; \\
e_2 & \quad x_1 \times x_2 = e = 0;
\end{align*}
\]

will give the message

```
x2 appearing in
  e2: Pivot too small for triangular model. Value=0.000E+00
```

** Infeasible solution. The equations were assumed to be recursive but they are not. A pivot element is too small.

However, the uniqueness of \(x_2\) may not be relevant if the solution just is going to be used as an initial point for a second model. The option `lsismp = t` (for Logical Switch: Ignore Small Pivots) will allow zero pivots as long as the corresponding equation is feasible for the given initial values.

### 5.7.8.13.2 Constrained Nonlinear System or Square Systems of Equations

There is a special model class in GAMS called CNS - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e. a model in which the number of non-fixed variables is equal to the number of constraints. Currently, CONOPT and PATH are the only solvers for this model class. A CNS model can be solved with a solve statement like

```
Solve <model> using CNS;
```

without an objective term. In some cases it may be convenient to solve a CNS model with a standard solve statement combined with an options file that has the statement `lssqrs = t`. In the latter case, CONOPT will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. The solution process does not include a lot of the safeguards used for ordinary NLP models and when it work it is often very fast and it uses less memory than for the corresponding NLP model. The lack of safeguards means that the solution process just will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Examples of difficulties are that the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds as described with examples below.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

The solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is

```
** Feasible solution to a square system.
```

that usually will be combined with model status 16-Solved. If CONOPT in special cases can guarantee that the solution is unique, for example if the model is linear, then the model status will be 15-Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints
5.7 CONOPT

\begin{verbatim}
e1 .. x1 + x2 =e= 1;
e2 .. 2*x1 + 2*x2 =e= 2;
\end{verbatim}

will result in the message

** Error in Square System: Pivot too small.
   e2: Pivot too small.
   x1: Pivot too small.

"Pivot too small" means that the set of constraints is linearly dependent in the current point and there is no unique search direction for Newton's method so CONOPT terminates. The message points to one variable and one constraint. However, this just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error will usually be combined with model status 5 - Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If the constraints are linearly dependent but the current point satisfy the constraints then the solution status will be 17 - Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

It should be mentioned that the linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is

\begin{verbatim}
e1 .. x1*x2 =E= 1;
e2 .. x1+x2 =E= 3;
x1.l = 1; x2.l = 1;
\end{verbatim}

A model with these two constraints and the bound

\begin{verbatim}
e1 .. x1 + x2 =e= 2;
e2 .. x1 - x2 =e= 0;
x1.lo = 1.5;
\end{verbatim}

will result in the message

** Error in Square System: A variable tries to exceed its bound.
   x1: The variable tries to exceed its bound.

because the solution, \((x1,x2) = (1,1)\) violates the bound on \(x1\). This error case will also be combined with model status 5-Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If you encounter problems with active bounds but you think it is caused by nonlinearities and that there is a solution, then you may try to use the bending linesearch with option \textit{lmmxsfs} = t.

The CNS facility can be used to generate an initial feasible solution in almost the same way as the triangular model facility: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the CNS solver or with \textit{1ssqrs} = t, reset the bounds on the fixed variables, and solve the original model. The CNS facility can be used on a larger class of models that include simultaneous sets of equations. However, the square system must be non-singular and feasible; CONOPT cannot, like in the triangular case, add artificial variables to some of the constraints and solve the remaining system when a variable reaches one of its bounds.

Additional information on CNS can be found in the GAMS User's Guide.
5.7.8.13.3 Loss of Feasibility  During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message ”Loss of Feasibility - Return to Phase 0”. The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization where the objective changes rapidly fairly large infeasibilities may be acceptable. As the change in objective in each iteration becomes smaller it will be necessary to solve the constraints more accurately so the ”noise” in objective value from the inaccurate constraints will remain smaller than the real change. The noise is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g. when an inequality becomes binding. In these cases CONOPT will tighten the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick and it happens silently. However, Newton's method may fail, for example in cases where the model is degenerate and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section Finding a Feasible Solution: Phase 0 and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the ”Loss of feasibility ...” message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.

In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. We are working on reducing this problem. Until a final solution has been implemented you are encouraged to (1) consider if bounds on some degenerate variables can be removed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, rtwma and rtwmi (see Appendix B), if this happens with your model.

5.7.8.13.4 Stalling  CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly.

The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus a noise adjustment term equal to the scalar product of the residuals with the marginals (see section Loss of Feasibility where this noise term also is used). The noise adjustment term is very useful in allowing CONOPT to work smoothly with fairly inaccurate intermediate solutions. However, there is a disadvantage: the noise adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT loses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To avoid infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see OK = F in section Linear and Nonlinear Mode: Phase 1 to 4). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds lfstal and lfstal is positive. The default value of lfstal is 100. The message will be:

** Feasible solution. The tolerances are minimal and there is no change in objective although the reduced gradient is greater than the tolerance.

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum fairly accurately then you may have to increase the value of lfstal.
5.7.8.13.5 Overflow and NaN (Not A Number)  Very large values or overflow can appear in a number of places. CONOPT cannot use these values in the optimization since the results would be unreliable or useless. CONOPT is therefore trying to detect these values and take appropriate action.

Even though most modern computers can work with numbers from 1.e-300 to 1.e+300 CONOPT considers all numbers that are larger than 1.e40 or NaN to be 'bad' and useless. NaN means Not A Number and includes Infinity and Real Overflow.

Bad numbers can have two sources. They can come from the modeler expressions or they can be generated internally in CONOPT. The nonlinear constraints or the derivatives of these constraints may return very large values. If a constraint returns a bad number you will see a message like

** A function value is very large or NaN (Not a Number).
Add bound to ensure that the function values are defined and bounded.

and if a derivatives returns a bad number the message will be something like

** A derivative is very large or NaN (Not a Number).
Add bound to ensure that the derivatives are defined and bounded.

The modeler must in both cases adjust the model to prevent that CONOPT receives these bad numbers so CONOPT stops immediately.

Bad numbers can also in rare cases appear as a result of computations inside CONOPT. Since all primal variables and all derivatives are bounded the bad numbers can only appear as a result of operations involving the factorization of the basis matrix, including the dual variables. You will in this case see a message like

** Overflow or Nan (Not A Number) has been encountered indicating numerical difficulties. Trying to repair by increasing the absolute and relative pivot tolerances (Rtpiva and Rvpivr) and allowing small values to be scaled up by decreasing the minimum scale factor, Rtmins.

Since the source of the problem is the factorization of the basis CONOPT will adjust the tolerances that are used while computing this factorization. Essentially, we must avoid very small pivots and the pivoting tolerances are therefore increased. The source of very small pivots can also be constraints were all terms are very small so CONOPT will also change the minimum scaling factor to try to avoid this source. CONOPT will continue the optimization after changing the tolerances and if the problem appears again if will change the tolerances even more, but since there is a limit to how much they can be adjusted CONOPT may have to give up and it will happen with this message

** Overflow or Nan (Not A Number) continues to be encountered after multiple attempt to repair by changed tolerances. CONOPT will give up.

The termination message will in this case be

** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).

or
** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).

If you encounter messages with NaN it is always a problem with scaling or structure. The most likely source is constraints with only small terms. Try to avoid them by scaling the variables and constraints appropriately, or try to use option `rtmins`. Or make CONOPT ignore these constraints by using a larger absolute pivot tolerance, `rtpiva`.

Another source are structured models with long chains of relationships. These models will usually have lags or leads and neighboring variables are related with a factor different from one, i.e.

```plaintext
dyn(i) .. x(i) =e= x(i-1)*1.5 + u(i);
```

or

```plaintext
dif(i) .. 2*x(i) =e= x(i-1) + x(i+1) + u(i);
```

CONOPT will try to solve the constraints in the proper order to avoid exploding variables, but a larger relative pivot tolerance, `rtpivr`, may also help. If you have a dynamic model and the variables are supposed to grow rapidly over time then you may consider having a scale factor that also grows with time.

### 5.7.8.13.6 External Equations and Extrinsic Functions

CONOPT can be used with external equations and extrinsic functions written in a programming language such as Fortran or C. Additional information is available in the GAMS User's Guide "External Equations and Extrinsic Functions."

Extrinsic functions can be debugged using the function suffixes `gradn` and `hessn` which use finite differences to approximate the gradient and Hessian using multiple function calls. For details check model `drivtst` in the GAMS Model Library. If external equations are present CONOPT will automatically turn the Function and Derivative Debugger on in the initial point to discover potential errors in the gradient calculation inside the external library. The debugger will not only check the gradients of the external library but even check the gradients calculated by GAMS. In rare cases the debugger may report problems in the regular algebra part of the model for which GAMS has calculated the gradients. These violates should be small and can be ignored. After verifying that the external part of the model has been programmed correctly you may turn debugging off again by setting `lkdebg` to 0 in an options file.

The debugger has two types of check. The first type ensures that the external equations do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these check are usually caused by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external equations and extrinsic functions are consistent with the rate of change in function values. A derivative is considered to be wrong if the value returned by the modeler deviates from the value computed using numerical differences by more than `rtmxj2` times the step used for the numerical difference (usually around 1.e-7). This check is correct if second derivatives are less than `rtmxj2`. `rtmxj2` has a default value of 1.e4. If your model has larger second derivatives you may increase it in order not to get wrong error messages.

The number of error messages from the Function and Derivative Debugger is limited by `lfderr` with a default value of 10.

See Debugging options for a list of options that control the debugger.

### 5.7.9 Appendix B - Options

The options that ordinary GAMS users can access are listed below. Options starting on R assume real values, options starting on LS assume logical values (TRUE, T, 1, or FALSE, F, or 0), and all other CR-Cells starting on L assume integer values. The logical option `dohess` is only used by the interface between GAMS and CONOPT.

#### 5.7.9.1 Algorithmic options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF2DRV</td>
<td>Limit on errors in Directional Second Derivative evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>LFDEGI</td>
<td>Limit on number of degenerate iterations before starting degeneracy breaking strategy.</td>
<td>10</td>
</tr>
<tr>
<td>LFEERR</td>
<td>Limit on number of function evaluation errors. Overwrites GAMS Domlim option</td>
<td>GAMS DomLim</td>
</tr>
<tr>
<td>LFHSOK</td>
<td>Limit on errors in Hessian evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>LFITER</td>
<td>Maximum number of iterations. Overwrites GAMS Iterlim option.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>LFMXNS</td>
<td>Maximum number of new superbasic variables added in one iteration.</td>
<td>auto</td>
</tr>
<tr>
<td>LFNICR</td>
<td>Limit on number of iterations with slow progress (relative less than Rtobjl).</td>
<td>20</td>
</tr>
<tr>
<td>LFNSUP</td>
<td>Maximum number of superbasic variables in the approximation to the Reduced Hessian.</td>
<td>auto</td>
</tr>
<tr>
<td>LFSCAL</td>
<td>Rescaling frequency.</td>
<td>5</td>
</tr>
<tr>
<td>LFSTAL</td>
<td>Limit on the number of stalled iterations.</td>
<td>100</td>
</tr>
<tr>
<td>LFUSDF</td>
<td>Limit on the number of candidates for defined variable in one constraint</td>
<td>2</td>
</tr>
<tr>
<td>LMDEBG</td>
<td>Method used by the function and derivative debugger.</td>
<td>0</td>
</tr>
<tr>
<td>LMMXSF</td>
<td>Method used to determine the step in Phase 0.</td>
<td>0</td>
</tr>
<tr>
<td>LMMXST</td>
<td>Method used to determine the maximum step while tightening tolerances.</td>
<td>0</td>
</tr>
<tr>
<td>LMNDIA</td>
<td>Method for initializing the diagonal of the approximate Reduced Hessian</td>
<td>0</td>
</tr>
<tr>
<td>LMSCAL</td>
<td>Method used for scaling.</td>
<td>3</td>
</tr>
<tr>
<td>LMUSDF</td>
<td>Method used with defined variables</td>
<td>0</td>
</tr>
<tr>
<td>LS2NDI</td>
<td>Flag for approximating Hessian information for incoming superbasics.</td>
<td>0</td>
</tr>
<tr>
<td>LS2PTJ</td>
<td>Flag for use of perturbations to compute 2nd derivatives in SQP method.</td>
<td>1</td>
</tr>
<tr>
<td>LSANRM</td>
<td>Flag for turning Steepest Edge on.</td>
<td>0</td>
</tr>
<tr>
<td>LSCRSH</td>
<td>Flag for crashing an initial basis without fixed slacks.</td>
<td>1</td>
</tr>
<tr>
<td>LSESLP</td>
<td>Flag for enabling SLP mode.</td>
<td>1</td>
</tr>
<tr>
<td>LSESQP</td>
<td>Flag for enabling of SQP mode.</td>
<td>1</td>
</tr>
<tr>
<td>LSISMP</td>
<td>Flag for Ignoring Small Pivots in Triangular models</td>
<td>0</td>
</tr>
<tr>
<td>LSLACK</td>
<td>Flag for selecting initial basis as Crash-triangular variables plus slacks.</td>
<td>0</td>
</tr>
<tr>
<td>LSPOST</td>
<td>Pre-processor flag for identifying and using post-triangular equations.</td>
<td>1</td>
</tr>
<tr>
<td>LSPRET</td>
<td>Pre-processor flag for identifying and using pre-triangular equations.</td>
<td>1</td>
</tr>
<tr>
<td>LSSCAL</td>
<td>Flag for dynamic scaling.</td>
<td>1</td>
</tr>
<tr>
<td>LSSQRS</td>
<td>Flag for Square System. Alternative to defining modeltype=CNS in GAMS</td>
<td>0</td>
</tr>
<tr>
<td>LSTCRS</td>
<td>Flag for using the triangular crash method.</td>
<td>0</td>
</tr>
<tr>
<td>LSTRIA</td>
<td>Flag for triangular or recursive system of equations.</td>
<td>0</td>
</tr>
<tr>
<td>LSTRID</td>
<td>Flag for turning diagnostics on for the post-triangular equations.</td>
<td>0</td>
</tr>
<tr>
<td>LSUQDF</td>
<td>Flag for requiring defined variables to be unique</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>LSUSDF</td>
<td>Flag for forcing defined variables into the basis</td>
<td>1</td>
</tr>
<tr>
<td>PRDEF</td>
<td>Flag for printing the defined variables and their defining constraints.</td>
<td>0</td>
</tr>
<tr>
<td>PRPOST</td>
<td>Flag for printing the post-triangular part of the model</td>
<td>0</td>
</tr>
<tr>
<td>PRPREC</td>
<td>Flag for printing the variables changed by the pre-processor</td>
<td>0</td>
</tr>
<tr>
<td>PRPRET</td>
<td>Flag for printing the pre-triangular part of the model</td>
<td>0</td>
</tr>
<tr>
<td>RTBND1</td>
<td>Bound filter tolerance for solution values close to a bound.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTBNDT</td>
<td>Bound tolerance for defining variables as fixed.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTIPWA</td>
<td>Absolute Pivot Tolerance for building initial basis.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTIPVR</td>
<td>Relative Pivot Tolerance for building initial basis</td>
<td>1.e-3</td>
</tr>
<tr>
<td>RTMAXJ</td>
<td>Upper bound on the value of a function value or Jacobian element.</td>
<td>1.e10</td>
</tr>
<tr>
<td>RTMAXS</td>
<td>Upper bound on scale factors.</td>
<td>1.e9</td>
</tr>
<tr>
<td>RTMAXV</td>
<td>Upper bound on solution values and equation activity levels</td>
<td>1.e10</td>
</tr>
<tr>
<td>RTMINA</td>
<td>Zero filter for Jacobian elements and inversion results.</td>
<td>1.e-20</td>
</tr>
<tr>
<td>RTMINJ</td>
<td>Filter for small Jacobian elements to be ignored during scaling.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>RTMINS</td>
<td>Lower bound for scale factors computed from values and 1st derivatives.</td>
<td>1</td>
</tr>
<tr>
<td>RTMNS2</td>
<td>Lower bound for scale factors based on large 2nd derivatives.</td>
<td>1.e-6</td>
</tr>
<tr>
<td>RTNWMA</td>
<td>Maximum feasibility tolerance (after scaling).</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTNWMI</td>
<td>Minimum feasibility tolerance (after scaling).</td>
<td>4.e-10</td>
</tr>
<tr>
<td>RTNWTR</td>
<td>Feasibility tolerance for triangular equations.</td>
<td>2.0e-8</td>
</tr>
<tr>
<td>RTOBJL</td>
<td>Limit for relative change in objective for well-behaved iterations.</td>
<td>3.0e-12</td>
</tr>
<tr>
<td>RTOBJR</td>
<td>Relative accuracy of the objective function.</td>
<td>3.0e-13</td>
</tr>
<tr>
<td>RTONED</td>
<td>Accuracy of One-dimensional search.</td>
<td>0.2</td>
</tr>
<tr>
<td>RTPIVA</td>
<td>Absolute pivot tolerance.</td>
<td>1.e-10</td>
</tr>
<tr>
<td>RTPIVR</td>
<td>Relative pivot tolerance during basis factorizations.</td>
<td>0.05</td>
</tr>
<tr>
<td>RTPIVT</td>
<td>Absolute pivot tolerance for nonlinear elements in pre-triangular equations.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>RTPIVU</td>
<td>Relative pivot tolerance during basis updates.</td>
<td>0.05</td>
</tr>
<tr>
<td>RTPREC</td>
<td>Tolerance for printing variables changed by the pre-processor</td>
<td>0.0</td>
</tr>
<tr>
<td>RTPREL</td>
<td>Tolerance for defining large changes in variables in pre-processor</td>
<td>0.01</td>
</tr>
<tr>
<td>RTREDG</td>
<td>Optimality tolerance for reduced gradient.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RVFILL</td>
<td>Fill in factor for basis factorization.</td>
<td>5</td>
</tr>
<tr>
<td>RVTIME</td>
<td>Time Limit. Overwrites the GAMS Reslim option.</td>
<td>GAMS ResLim</td>
</tr>
</tbody>
</table>

### 5.7.9.2 Debugging options

...
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFDERR</td>
<td>Limit on number of error messages from function and derivative debugger.</td>
<td>10</td>
</tr>
<tr>
<td>LKDEBG</td>
<td>Flag for debugging of first derivatives</td>
<td>0</td>
</tr>
<tr>
<td>RTMXJ2</td>
<td>Upper bound on second order terms.</td>
<td>1.e4</td>
</tr>
<tr>
<td>RTZERN</td>
<td>Zero-Noise in external equations</td>
<td>0.0</td>
</tr>
</tbody>
</table>

5.7.9.3 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFEMSG</td>
<td>Limit on number of error messages related to large function value and Jacobian elements.</td>
<td>10</td>
</tr>
<tr>
<td>LFILOG</td>
<td>Frequency for log-lines for non-SLP/SQP iterations.</td>
<td>auto</td>
</tr>
<tr>
<td>LFILOS</td>
<td>Frequency for log-lines for SLP or SQP iterations.</td>
<td>auto</td>
</tr>
<tr>
<td>LFTMSG</td>
<td>Limit on number of error messages related to infeasible pre-triangle.</td>
<td>25</td>
</tr>
</tbody>
</table>

5.7.9.4 Interface options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooptfile</td>
<td>Flag for computing and using directional 2nd derivatives.</td>
<td>auto</td>
</tr>
<tr>
<td>DO2DIR</td>
<td>Flag for computing and using 2nd derivatives as Hessian of Lagrangian.</td>
<td>auto</td>
</tr>
<tr>
<td>heaplimit</td>
<td>Maximum Heap size in MB allowed</td>
<td>Infinite</td>
</tr>
<tr>
<td>pretri2log</td>
<td>Send messages about the pre-triangular analyser to the log</td>
<td>0</td>
</tr>
<tr>
<td>RVHESS</td>
<td>Memory factor for Hessian generation: Skip if #Hessian elements &gt; #Jacobian elements∗Rvhess, 0 means unlimited.</td>
<td>10</td>
</tr>
</tbody>
</table>

cooptfile (string): ←

DO2DIR (boolean): Flag for computing and using directional 2nd derivatives. ←

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off of the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see DOHESS) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: auto

DOHESS (boolean): Flag for computing and using 2nd derivatives as Hessian of Lagrangian. ←
If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with =X=) or if the Hessian becomes too dense. See also DO2DIR and RVHESS.

Default: auto

heaplimit (real): Maximum Heap size in MB allowed

Default: Infinite

LF2DRV (integer): Limit on errors in Directional Second Derivative evaluation.

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lf2drv times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: 10

LFDEGI (integer): Limit on number of degenerate iterations before starting degeneracy breaking strategy.

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds LFDEGI CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: 10

LFDERR (integer): Limit on number of error messages from function and derivative debugger.

The function and derivative debugger (see LKDEBG) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after LFDERR error have been found.

Range: \{1, ..., \infty\}

Default: 10

LFEERR (integer): Limit on number of function evaluation errors. Overwrites GAMS Domlim option

Synonym: domlim

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lfeerr times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: GAMS DomLim

LFEMSG (integer): Limit on number of error messages related to large function value and Jacobian elements.
Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function value and derivatives. This bound is $1\times10^30$ for scaled models and RTMAXJ for unscaled models. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating function value and Jacobian elements, up to a limit of Lfemsg error messages.

Range: $\{1, \ldots, \infty\}$

Default: 10

**LFHSOK** *(integer)*: Limit on errors in Hessian evaluation.

If the evaluation of Hessian information has failed more than Lfhsok times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: 10

**LFILOG** *(integer)*: Frequency for log-lines for non-SLP/SQP iterations.

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Range: $\{1, \ldots, \infty\}$

Default: auto

**LFILOS** *(integer)*: Frequency for log-lines for SLP or SQP iterations.

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

Range: $\{1, \ldots, \infty\}$

Default: auto

**LFITER** *(integer)*: Maximum number of iterations. Overwrites GAMS Iterlim option.

Synonym: iterlim

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option RVTIME) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: GAMS IterLim
**LFXMNS** *(integer)*: Maximum number of new superbasic variables added in one iteration. 

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lfxmns is positive then the limit is min(500,Lfxmns). If Lfxmns is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: **auto**

**LFNICR** *(integer)*: Limit on number of iterations with slow progress (relative less than **RTOBJL**).

The optimization is stopped if the relative change in objective is less than **RTOBJL** for Lfnicr consecutive well-behaved iterations.

Range: \{2, ..., ∞\}

Default: 20

**LFNSUP** *(integer)*: Maximum number of superbasic variables in the approximation to the Reduced Hessian.

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on on the size. The limit is LFNSUP if LFNSUP is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Range: \{5, ..., ∞\}

Default: **auto**

**LFSCAL** *(integer)*: Rescaling frequency.

The row and column scales are recalculated at least every Lfscal new point (degenerate iterations do not count), or more frequently if conditions require it.

Range: \{1, ..., ∞\}

Default: 5

**LFSTAL** *(integer)*: Limit on the number of stalled iterations.

An iteration is considered a stalled iteration if there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will usually try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, so to prevent cycling CONOPT stops after Lfstal stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Range: \{2, ..., ∞\}

Default: 100
**LFTMSG (integer):** Limit on number of error messages related to infeasible pre-triangle.

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than LFTMSG elements the report is considered difficult to use and it is skipped.

Default: 25

**LFUSDF (integer):** Limit on the number of candidates for defined variable in one constraint

When there are more than one candidate to be selected as a defined variable in a constraint CONOPT tries to select the most appropriate in order to select as many defined variables as possible. However, to avoid too much arbitrariness this is only attempted if there are not more than Lfusdf candidates.

Default: 2

**LKDEBG (integer):** Flag for debugging of first derivatives

Lkdebg controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with =X=. The amount of debugging is controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The derivatives are tested in the initial point only.</td>
</tr>
<tr>
<td>0</td>
<td>No debugging</td>
</tr>
<tr>
<td>+n</td>
<td>The derivatives are tested in all iterations that can be divided by Lkdebg, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)</td>
</tr>
</tbody>
</table>

**LMDEBG (integer):** Method used by the function and derivative debugger.

The function and derivative debugger (turned on with LKDEBG) can perform a fairly cheap test or a more extensive test, controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.</td>
</tr>
<tr>
<td>1</td>
<td>As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used of the cheap test does not find an error but one is expected to exist.</td>
</tr>
</tbody>
</table>

**LMMXSF (integer):** Method used to determine the step in Phase 0.

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by LMMXSF:
Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.</td>
</tr>
<tr>
<td>1</td>
<td>A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending. The method does not use anti-degeneracy. This will to be taken care off by removing difficult constraints from the Newton Process at regular intervals. The bending method can sometimes be useful for CNS models that stop when a variable exceeds its bound in an intermediate point even though the final solution is known to be inside the bounds.</td>
</tr>
</tbody>
</table>

**LMMXST (integer):** Method used to determine the maximum step while tightening tolerances.

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by LMMXST:

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.</td>
</tr>
<tr>
<td>1</td>
<td>A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.</td>
</tr>
</tbody>
</table>

**LMNDIA (integer):** Method for initializing the diagonal of the approximate Reduced Hessian

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by LMNDIA:

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.</td>
</tr>
<tr>
<td>1</td>
<td>The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.</td>
</tr>
</tbody>
</table>

**LMSCAL (integer):** Method used for scaling.

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution, see
LSSCAL and LMSCAL. The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see RTMAXS and RTMINS. Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than RTMINJ are considered equal to RTMINJ.</td>
</tr>
<tr>
<td>1</td>
<td>Similar to 3 below, but the projection on the interval [Rtmins,Rtmaxs] is applied at a different stage. With method 1, abs(X)∗abs(Jac) with small X and very large Jac is scaled very aggressively with a factor abs(Jac). With method 3, the scale factor is abs(X)∗abs(Jac). The difference is seen in models with terms like Sqrt(X) close to X = 0.</td>
</tr>
<tr>
<td>2</td>
<td>As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.</td>
</tr>
<tr>
<td>3</td>
<td>Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by the maximum of the largest term and the value of the variable. A term is here defined as abs(X)∗abs(Jac) where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factors are then projected on the interval between Rtmins and Rtmaxs.</td>
</tr>
</tbody>
</table>

LMUSDF (integer): Method used with defined variables

When defined variables are identified (see LSUSDF) they can be used in two ways, controlled by LMUSDF:

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Defined variables are only used in the initial point and for the initial basis (default).</td>
</tr>
<tr>
<td>1</td>
<td>Defined variables are kept basic and the defining constraints are used to recursively assign values to the defined variables in all trial points.</td>
</tr>
</tbody>
</table>

LS2NDI (boolean): Flag for approximating Hessian information for incoming superbasics.

If LS2NDI is turned on (1) CONOPT will try to estimate second order Hessian information for incoming superbasic variables based on directional second derivatives. This is more costly than the standard method described under LMNDIA.

Default: 0

LS2PTJ (boolean): Flag for use of perturbations to compute 2nd derivatives in SQP method.

If on (1) CONOPT may use perturbations of the Jacobian to compute directional 2nd derivatives if they are not provided with other cheaper and more accurate methods. With GAMS it is only relevant for models with functions defined in external function libraries or models with external equations defined with the =X= equation type.

Default: 1
5.7 CONOPT

LSANRM (boolean): Flag for turning Steepest Edge on. ↔

Flag used to turn steepest edge pricing on (1) or off (0). Steepest edge pricing makes the individual iterations more expensive but it may decrease their number. Only experimentation can show if it is worth while.

Default: 0

LSCRSH (boolean): Flag for crashing an initial basis without fixed slacks ↔

When turned on (1) CONOPT will try to crash a basis without fixed slacks in the basis. Fixed slacks are only included in a last round to fill linearly dependent rows. When turned off, large infeasible slacks will be included in the initial basis with preference for variables and slacks far from bound.

Default: 1

LSESQP (boolean): Flag for enabling SQP mode. ↔

If Lsesqp is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: 1

LSISMP (boolean): Flag for Ignoring Small Pivots in Triangular models ↔

Ignore Small Pivots. When turned on CONOPT will ignore the non-uniqueness from small pivots during a triangular solve (see LSTRIA). Note, that the non-uniqueness may propagate to later equations, but we cannot check for it in nonlinear equations.

Default: 0

LSLACK (boolean): Flag for selecting initial basis as Crash-triangular variables plus slacks. ↔

When turned on together with LSTCRS CONOPT will use the triangular crash procedure and select the initial basis as the crash-triangular variables plus slacks in all remaining rows.

Default: 0

LSPOST (boolean): Pre-processor flag for identifying and using post-triangular equations. ↔

When turned on (the default) CONOPT will try to identify post-triangular equations. Otherwise this phase is ignored.

Default: 1

LSPRET (boolean): Pre-processor flag for identifying and using pre-triangular equations. ↔

When turned on (the default) CONOPT will try to identify pre-triangular equations. Otherwise this phase is ignored.

Default: 1
**LSSCAL (boolean):** Flag for dynamic scaling.

When Lsscal is on (the default) CONOPT will use dynamic scaling of equations and variables. See also **LMSCAL**.

Default: 1

**LSSQRS (boolean):** Flag for Square System. Alternative to defining modeltype=CNS in GAMS

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: 0

**LSTCRS (boolean):** Flag for using the triangular crash method.

When turned on CONOPT will try to crash a triangular basis using ideas by Gould and Reid. The procedure relies on identifying and using good initial values provided by the modeler and only assigning values to variables that are not initialized. Should only be used when several important variables have been given reasonable initial values. The sum of infeasibilities may for some models grow during the crash procedure, so modelers are advised that the option should be used with caution. The option will be ignored if defined variables are forced into the bases **Lsusdf**.

Default: 0

**LSTRIA (boolean):** Flag for triangular or recursive system of equations.

If turned on the equations must form a recursive system. Equations that depend on known variables are allowed as long as they are consistent, e.g. $x = 1$ and $2x = 2$. If the equations are not recursive the model is considered infeasible, and the equations with minimum row count are flagged together with the columns they intersect. See also **LSISMP**.

Default: 0

**LSTRID (boolean):** Flag for turning diagnostics on for the post-triangular equations.

If turned on certain diagnostic messages related to the post-triangular equations will be printed. The messages are mainly related to unusual modeling constructs where linear variables for example only appear in the objective or where certain constraints are guaranteed redundant.

Default: 0

**LSUQDF (boolean):** Flag for requiring defined variables to be unique

When turned on (1) CONOPT will not allow defined variables unless they are unique. We exclude a variable if it can be defined from more than one equation, and we exclude equations if they can be used to define more than one variable.

Default: 1

**LSUSDF (boolean):** Flag for forcing defined variables into the basis

When turned on (1) CONOPT will identify defined variables from constraints of the type $x(i) = f(x)$ where $x(i)$ is free or implied free. The largest number of defined variables possible will be made basic and will be assigned initial values that are consistent with their defining constraint. When turned off (0) defined variables and their defining constraints are treated like all other variables and constraints. When turned on the triangular crash (**LSTCRS**) will not be used.

Default: 1
**PRDEF** *(boolean)*: Flag for printing the defined variables and their defining constraints. →

When turned on (1) CONOPT will print a list of the defined variables and their defining constraints in the order in which they can be evaluated.

Default: 0

**pretri2log** *(boolean)*: Send messages about the pre-triangular analyser to the log →

Default: 0

**PRPOST** *(boolean)*: Flag for printing the post-triangular part of the model →

When turned on (1) CONOPT will print a list of the post-triangular constraints and the variables they are solved for in the order in which they can be evaluated.

Default: 0

**PRPREC** *(boolean)*: Flag for printing the variables changed by the pre-processor →

When turned on (1) CONOPT will print a list of the variables that are changed by the pre-processor. The list includes pre-triangular variables, variables changed by implied bounds, and definitional variables. The three lists can for some models be very long and you can limit the list to variables that are changed by more than (RTPREC)

Default: 0

**PRPRET** *(boolean)*: Flag for printing the pre-triangular part of the model →

When turned on (1) CONOPT will print a list of the pre-triangular constraints and the variables they are solved for, including the solution values, in the order in which they are solved.

Default: 0

**RTBND1** *(real)*: Bound filter tolerance for solution values close to a bound. →

A variable is considered to be at a bound if its distance from the bound is less than Rtbnd1 \( \text{Max}(1,\text{ABS}(\text{Bound})) \). If you need a very small value then your model is probably poorly scaled.

Range: \([3.e-13, 1.e-5]\)

Default: 1.e-7

**RTBNDT** *(real)*: Bound tolerance for defining variables as fixed. →

A variable is considered fixed if the distance between the bounds is less than Rtbndt \( * \text{Max}(1,\text{Abs}(\text{Bound})) \). The tolerance is also used on implied bounds (from converted inequalities) and these implied bounds may be infeasible up to Rtbndt.

Range: \([3.e-13, 1.e-5]\)

Default: 1.e-7

**RTIPVA** *(real)*: Absolute Pivot Tolerance for building initial basis. →
Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: \([3.\text{e}-13, 1.\text{e}-3]\)

Default: \(1.\text{e}-7\)

**RTIPVR (real):** Relative Pivot Tolerance for building initial basis

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: \([1.\text{e}-4, 0.9]\)

Default: \(1.\text{e}-3\)

**RTMAXJ (real):** Upper bound on the value of a function value or Jacobian element.

Very large values of variables, function value, and derivatives and in particular large variations in the absolute value of the variables, functions, and derivatives makes the model harder to solve and poses problems for both feasibility and optimality tests. CONOPT will usually try to scale the model (see LSSCAL) to remove these problems. However, scaling can also make important aspects of a model appear un-important and there is therefore a limit to how aggressively we can scale a model (see RTMAXS and RTMINS). To avoid serious scaling problems CONOPT poses upper bounds on all variables (see RTMAXV) and all function value and derivatives, RTMAXJ.

Range: \([1.\text{e}4, 1.\text{e}30]\)

Default: \(1.\text{e}10\)

**RTMAXS (real):** Upper bound on scale factors.

Scale factors are projected on the interval from Rtmins to Rtmaxs. Is used to prevent very large or very small scale factors due to pathological types of constraints. RTMAXS is silently increased to \(\text{max}(\text{RTMAXV,RTMAXS})/100\) if RTMAXV or RTMAXJ have large non-default values.

Range: \([1, 1.\text{e}20]\)

Default: \(1.\text{e}9\)

**RTMAXV (real):** Upper bound on solution values and equation activity levels

See RTMAXJ for a discussion of why CONOPT poses upper bounds on variables and derivatives. If the value of a variable, including the objective function value, exceeds RTMAXV then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded.

Range: \([1.\text{e}5, 1.\text{e}30]\)

Default: \(1.\text{e}10\)

**RTMINA (real):** Zero filter for Jacobian elements and inversion results.

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than RTPIVA/10.

Range: \([1.\text{e}-30, \infty]\)

Default: \(1.\text{e}-20\)
RTMINJ (real): Filter for small Jacobian elements to be ignored during scaling.

A Jacobian element is considered insignificant if it is less than $\text{Rtminj}$. The value is used to select which small values are scaled up during scaling of the Jacobian.

Range: $[1.e-7, 1.e-3]$  
Default: $1.e-5$

RTMINS (real): Lower bound for scale factors computed from values and 1st derivatives.

Scale factors used to scale variables and equations are projected on the range $\text{Rtmins}$ to $\text{Rtmaxs}$. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for $\text{Rtmins}$ is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of $\text{Rtmins} < 1$.

Range: $[1.e-10, 1]$  
Default: 1

RTMNS2 (real): Lower bound for scale factors based on large 2nd derivatives.

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than $\text{Rtmins}$. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than $\text{Rtmins2}$.

Range: $[1.e-9, 1]$  
Default: $1.e-6$

RTMXJ2 (real): Upper bound on second order terms.

The function and derivative debugger (see LKDEBG) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses $\text{RTMXJ2}$ as an upper bound on second order derivatives in the model. Larger $\text{RTMXJ2}$ values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Range: $[1, \infty]$  
Default: $1.e4$

RTNWMA (real): Maximum feasibility tolerance (after scaling).

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. $\text{RTNWMA}$ is the upper bound on the dynamic feasibility tolerance and $\text{RTNWMI}$ is the lower bound.

Range: $[1.e-10, 1.e-3]$  
Default: $1.e-7$

RTNWMI (real): Minimum feasibility tolerance (after scaling).
See RTNWMA for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: [4.e-11, 1.e-5]

Default: 4.e-10

RTNWTR (real): Feasibility tolerance for triangular equations.

Triangular equations are usually solved to an accuracy of RTNWMI. However, if a variable reaches a bound or a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to Rtnwtr.

Range: [3.e-13, 1.e-4]

Default: 2.0e-8

RTOBJL (real): Limit for relative change in objective for well-behaved iterations.

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than Rtobjl * Max(1,Abs(Objective)). See also LFNICR.

Range: [3.0e-13, 1.0e-5]

Default: 3.0e-12

RTOBJR (real): Relative accuracy of the objective function.

It is assumed that the objective function can be computed to an accuracy of Rtobjr * max( 1, abs(Objective) ). Smaller changes in objective are considered to be caused by round-off errors.

Range: [3.0e-14, 10.e-6]

Default: 3.0e-13

RTONED (real): Accuracy of One-dimensional search.

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than Rtoned times the decrease so far in this onedimensional search.

Range: [0.05, 0.8]

Default: 0.2

RTPIVA (real): Absolute pivot tolerance.

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Rtpiva. There is also a relative test, see RTPIVR.

Range: [2.2e-16, 1.e-7]

Default: 1.e-10

RTPIVR (real): Relative pivot tolerance during basis factorizations.
During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least \( R_{tpivr} \) * the largest absolute value in the column. Small values or \( R_{tpivr} \) will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of \( R_{tpivr} \) may help.

Range: \([1.0e-3, 0.9]\]

Default: 0.05

\textbf{RTPIVT (real)}: Absolute pivot tolerance for nonlinear elements in pre-triangular equations. \( \leftarrow \)

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is \( R_{tpiva} \). The value cannot be less that \( R_{tpiva} \).

Range: \([2.2e-16, 1.0e-3]\]

Default: \( 1.0e-5 \)

\textbf{RTPIVU (real)}: Relative pivot tolerance during basis updates. \( \leftarrow \)

During basis changes CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least \( R_{tpivu} \) * the other element. Smaller values of \( R_{tpivu} \) will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: \([1.0e-3, 0.9]\]

Default: 0.05

\textbf{RTPREC (real)}: Tolerance for printing variables changed by the pre-processor \( \leftarrow \)

The list of variables changed by the pre-processor (see (PRPREC)) can be limited to the variables that are changed by more than \( RTPREC \), defined as \( \text{deltaX} > RTPREC \times \max(1, \text{abs}(X)) \).

Range: \([0.0, \infty]\)

Default: 0.0

\textbf{RTPREL (real)}: Tolerance for defining large changes in variables in pre-processor \( \leftarrow \)

The statistics for the number of variables that are changed in the pro-processor are classified as small and large changes. The change, \( \text{DeltaX} \), is defined as large if \( \text{deltaX} > RTPREL \times \max(1, \text{abs}(X)) \).

Range: \([0.0, \infty]\)

Default: 0.01

\textbf{RTREDG (real)}: Optimality tolerance for reduced gradient. \( \leftarrow \)
The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than $R_{\text{tredg}}$.

Range: $[3.\text{e-13}, 1]$

Default: $1.\text{e-7}$

**RTZERN** *(real): Zero-Noise in external equations ←*

By default CONOPT will debug the derivatives returned from External functions in GAMS in the initial point (see **LKDEBG**). If external functions have constant derivatives then the constant terms are still part of the external function and this can give rise to small inaccuracies in the contribution of the constant derivatives to the function value. This noise can cause the debugger to incorrectly state that a the external equation depend on the variable with the constant derivative. A larger value of Rtzern should remove the error.

Range: $[0.0, 1]$

Default: 0.0

**RVFILL** *(real): Fill in factor for basis factorization. ←*

Rvfill is used in the initial allocation of memory for the factorization of the basis. The fill-in (number of new nonzeros) is assumed to be Rvfill-1 times the initial number of nonzeros in the basis. The default is 5 but you may experiment with a smaller value (down to 1.0) for models that use too much memory to get started. If Rvfill is small you may get slower execution due to increased memory movement. And you may still run out of memory later in the optimization.

Range: $[1.00, 20]$

Default: 5

**RVHESS** *(real): Memory factor for Hessian generation: Skip if $\#\text{Hessian elements} > \#\text{Jacobian elements} \times \text{Rvhess}$, 0 means unlimited. ←

The Hessian of the Lagrangian is considered too dense and is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by Rvhess. The assumption is that a very dense Hessian is expensive both to compute and use. If Rvhess = 0.0 then there is no limit on the number of Hessian elements.

Default: 10

**RVTIME** *(real): Time Limit. Overwrites the GAMS Reslim option. ←*

Synonym: reslim

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Rvtime is overwritten by Reslim when called from GAMS. Rvtime is defined in ProbSize and/or UpdtSize when used as a subroutine library.

Default: GAMS ResLim
5.7.10 APPENDIX C: References


5.8 CONOPT4

Author
Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.8.1 Introduction

CONOPT4 is an NLP solver derived from CONOPT3 but with many changes. This initial note will describe some of the more important changes from CONOPT3 to CONOPT4 and some of the new options that control these changes. Selected parts of the log-file are also shown and explained. The note is fairly technical and the casual user does not have to understand the details.

More self-contained documentation for CONOPT4 without numerous references to CONOPT3 will appear later.

5.8.2 When should you use CONOPT4?

CONOPT3 is a mature solver with a lot of build-in heuristics developed over years of experimentation and CONOPT3 works well for a large class of models. Initially, CONOPT4 is being developed and tuned for models where CONOPT3 has problems or where small adjustments have given a better solver. Our initial recommendations are:

- CONOPT4 should be tried on models that take more than a few minutes to solve. CONOPT3 is probably your best guess for small and medium sized models, i.e. models with less than 1,000 to 10,000 variables and constraints.
- CONOPT4 should be tried for large CNS models, i.e. models with more than 100,000 variables and constraints.
- CONOPT4 should be tried for models where CONOPT3 runs out of memory. However, note that models of this size under all circumstances are very large and good behavior cannot be guaranteed.
- CONOPT4 should be tried on models where CONOPT3 ends in a locally infeasible solution. CONOPT4 has several new components that avoid obviously infeasible areas and let CONOPT4 move away from saddle points.
- CONOPT4 should be tried on models where CONOPT3 finds a large number of definitional constraints.
5.8.3 Memory Management

CONOPT3 has a limit on the amount of memory that can be used of 8 GBytes. The way memory is allocated and used has been completely rewritten and there is no longer a logical limit in CONOPT4.

5.8.4 Revised Preprocessor

The preprocessor in CONOPT3 identifies pre- and post-triangular variables and constraints, and it handles these variables and constraints in a special way so some internal routines can run more efficiently. The triangular variables are for example always basic. CONOPT3 does not include these variables in tests for entering and leaving the basis. And triangular variables are processed before other variables in the inversion routine. Despite the special status of some variables and constraints, they are all part of the model to be solved.

CONOPT4 distinguished between a 'user model' as defined by the user via the GAMS language, and an 'internal model'. Pre-triangular variables and constraints are simply removed from the user model. Post-triangular variables and constraints are collapsed into a single condensed objective function. And definitional constraints are eliminated. After the internal model has been solved CONOPT4 translates the internal solution back into the solution for the user model and reports this solution to the user. Because CONOPT4 uses these two different models it is no longer possible to turn the preprocessor off.

In addition to the simple pre- and post-triangular variables and constraints from CONOPT3, the preprocessor in CONOPT4 looks at more possibilities for simplifying the model. Some of the new features are:

- Fixed variables are removed completely.
- Constraints that represent simple inequalities are identified and changed into simple bounds on the variables and the constraints are removed.
- Simple monotone constraints such as \( \exp(x) \leq c1 \) or \( \log(y) \leq c2 \) are converted into simple bounds on the variables and then removed.
- Forcing constraints such as \( x_1 + x_2 \leq 0 \) with \( x_1.lo = 0 \) and \( x_2.lo = 0 \) are identified, the variables are fixed, and the constraints are removed. If a forcing constraint is identified then other constraints may become pre-triangular so they also can be removed.
- Linear and monotone constraints are used to compute 'implied bounds' on many variables and these bounds can help CONOPT4 get a better starting point for finding an initial feasible solution.
- Some non-monotone constraints such as \( \text{sqr}(x_1) + \text{sqr}(x_2) \leq 1 \) can also be used to derive implied bounds (here \(-1 < x_1 < +1 \text{ and } -1 < x_2 < +1\)) that both can improve the starting point and can be used to determine that other terms are monotone.
- Constraints with exactly two variables, e.g. simple linear identities such as \( x_1 = a \times x_2 + b \) or simple monotone identities such as \( x_3 = \exp(x_4) \), are used to move bounds between the two variables and this may result in more variables being included in the post-triangle.
- Linear constraints that are identical or proportional to others are identified and removed.
- Pairs of constraints that define a lower and an upper bound on the same linear expression or proportional linear expressions, e.g. \( 1 \leq x_1 + x_2 \) and \( 2x_1 + 2x_2 \leq 4 \), are turned into a single ranged constraint implemented with a double-bounded slack variable.
- Nonlinear constraints that become linear when the pre-triangular variables are fixed are recognized as being linear with the resulting simplifications.
Some of the new preprocessing steps are useful when solving sub-models in a Branch and Bound environment. A constraint like \( x \leq M y \) where \( y \) is a binary variable fixed at either 0 or 1 is turned into a simple bound on \( x \). And a constraint like \( \sum(i, x(i)) \leq \text{Cap} \times y \) (with \( x\.lo(i) = 0 \)) combined with \( y \) fixed at zero will force all \( x(i) \) to zero.

The preprocessor also identifies constructs that are easy to make feasible. There are currently two types:

- **Penalty terms:** We define a penalty constraint as a constraint of the form \( f(x_1, x_2, \ldots) + p - n = 0 \), where \( p \) and \( n \) are positive variables, and where \( p \) and \( n \) only appear in post-triangular constraints or in previously identified penalty constraint. For any feasible values of the \( x \)-variables it is easy to find values of \( p \) and \( n \) that makes the penalty constraint feasible: \( p = \max(0, -f(x)) \) and \( n = \max(0, f(x)) \). The definition is easily generalized to constraints where \( p \) and \( n \) have coefficients different from one and nonzero bounds; the essence is the presence of two linear unbounded terms of opposite sign.

- **Minimax terms:** We define a minimax group as a group of constraints of the form \( e_{q(i)}..fi(x_1, x_2, \ldots) \leq z \) where \( z \) is common to the group and otherwise only appear in post-triangular constraints, and \( z \) is unbounded from above. For any feasible value of the \( x \)-variables it is easy to find a value of \( z \) that makes the minimax group feasible: \( z = \text{smin}(i: fi(x)) \). The definition is easily generalized to groups of constraints where \( z \) has coefficients different from one and where the direction of the inequality is reversed.

The preprocessor will also recognize definitional equations: constraints of the form \( x = f(y) \), where \( x \) is a free variable or the bounds on \( x \) cannot be binding, are called definitional equations and \( x \) is called a defined variable. If there are many potential defined variable the preprocessor will select a recursive set and logically eliminate them from the internal model: The values of the defined variables are easily derived from the values of all other variables by evaluating the definitional equations in their recursive order, and these values are substituted into the remaining constraints before their residuals are computed. The matrix of derivatives of the remaining constraints is computed from the overall matrix of derivatives via an elimination of the triangular definitional equations.

The procedure used to recognize definitional equation is similar to the one used in CONOPT3. However, the actual use of the definitional equations is quite different in CONOPT4. The definitional equations are eliminated from the internal model and they are not present in the internal operations used to solve this model. For some models this can make a big difference.

We will show a few examples of log files where the output from the preprocessor is shown. The first is from the `otpop.gms` model in the GAMS Library:

```
CONOPT version 4.02
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark
Licensed to: GAMS/CONOPT OEM License

The user model has 104 variables and 77 constraints
with 285 Jacobian elements, 100 of which are nonlinear.
The Hessian of the Lagrangian has 17 elements on the diagonal,
33 elements below the diagonal, and 66 nonlinear variables.

Iter  Phase  Ninf  Infeasibility  RGmax  NSB  Step  InItr  MX  OK
0     0     4.0939901439E+03 (Input point)

The pre-triangular part of the model has 32 constraints and 43 variables.
The post-triangular part of the model has 9 constraints and variables.
There are 13 definitional constraints and defined variables.
```
Preprocessed model has 39 variables and 23 constraints with 88 Jacobian elements, 22 of which are nonlinear.

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0510968588E+03 (Full preprocessed model)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.5063072017E+01 (After scaling)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0023365462E+00 (After adjusting individual variables)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first few lines define the size of the user model. Usually there will also be information on the Hessian of the Lagrangian; these two lines are missing from models where the Hessian is not generated, usually because it is very large and dense (or because the generation has been turned off with the option Flg.Hessian = false).

After the description of the initial sum of infeasibilities there are three lines with statistics from the preprocessor. Note that the pre-triangular part of the model has more variables than constraints, in this case because there are fixed variables. The post-triangular part will always have the same number of constraints as variables, and the same is true for definitional constraints and defined variables.

After the statistics from the preprocessor the size of the resulting internal model is shown. There is no Hessian information; it is costly to derive the Hessian for the internal model and it will in most cases be very dense so all use of 2nd order information in the internal model is computed by mapping data to the user model and using the real Hessian.

The next lines show the change in the sum of infeasibilities during the initial stages. The first line is after the preprocessor has changed some variables and removed many constraints. The second line is after the internal model has been scaled. And last line is after some variables have been adjusted as described in the next section.

The second example is from the prolog.gms model in the GAMS Library:

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>1.2977431398E+03 (Input point)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The post-triangular part of the model has 1 constraints and variables. There are 17 penalty and minimax constraints with 3 variables.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reduced model without penalty components has 17 variables and 5 constraints with 46 Jacobian elements, 0 of which are nonlinear.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.2960000000E+03 (Model without penalty constraints)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0625000000E+00 (After scaling)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0000000000E+00 (After adjusting individual variables)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are no pre-triangular variables and constraints and the pre-triangular line is therefore missing, just like the line describing definitional constraints. On the other hand, there is a line telling that the model has 17 penalty and minimax constraints involving a total of 3 variables.

The internal model that is generated is here without the penalty and minimax constraints. This particular sub-model and other internal sub-models are described in more detail later in this note.
5.8 CONOPT4

5.8.5 Phase 0 - Finding an Initial Feasible Solution

Phase 0 is started with a new 'Adjust Initial Point' procedure that tries to minimize the sum of infeasibilities by changing individual variables one at a time. The procedure is very cheap since each change of a single variable only involve a small part of the overall model, and it will as a by-product produce a large part of a good initial basis and many constraints will become feasible. As the log file examples above shows, the procedure can in some cases reduce the sum of infeasibilities significantly.

Phase 0 in CONOPT3 is based on Newton's method with a heuristic for taking some constraints out if they do not behave well for Newton. The method can be very fast if a good initial basis is found, but for larger models it can also use a large number of basis changes without any real improvement and can therefore be very slow. CONOPT4 has replaced the heuristic with a rigorous LP framework that iteratively finds a better basis for Newton's method. CONOPT4 can therefore be slower for very easy models but the variability in solution time is smaller and it should never be very slow.

The new Phase 0 procedure replaces the Triangular Crash procedure from CONOPT3.

The following example shows the relevant part of the log file for GAMS Library model otpop.gms:

```
<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.2657221820E+00</td>
<td>1.0E+00</td>
<td>3</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1.8510537501E+00</td>
<td>1.0E+00</td>
<td>2</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1.8243868306E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1.8239239188E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1.8239236521E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

** Feasible solution. Value of objective = 323.670194144

The InItr column shows the number of LP-like inner iterations for each outer iteration and Step = 1 indicates that the full solution from the LP was be used.

The next example is from GAMS Library model mathopt3.gms:

```
<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S0</td>
<td>0</td>
<td>2.6763223208E+00</td>
<td>1.2E-01</td>
<td>2</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1.3292743354E+00</td>
<td>1.0E+00</td>
<td>2</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>S0</td>
<td>0</td>
<td>4.9371333807E+00</td>
<td>2.5E-01</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>S0</td>
<td>0</td>
<td>2.3899063615E+00</td>
<td>1.0E+00</td>
<td>2</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S0</td>
<td>0</td>
<td>3.2207192474E+00</td>
<td>5.0E-01</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3.2206913182E+00</td>
<td>6.2E-02</td>
<td>1</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>6.8573593504E-01</td>
<td>1.7E+00</td>
<td>1</td>
<td>1.2E+00</td>
<td>1</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
```

For some of the iterations Step is less than 1 indicating that the direction found by the inner linear model could not be used in full due to nonlinearities. Also note the lines with 'S0' in the Phase column. The S tells that the model was scaled before the iteration and the sum of infeasibilities was increased during this scaling procedure. The sum of infeasibilities is therefore not monotone decreasing even if each outer iteration does decrease them.
5.8.6 Transition between SLP and SQP

The transition from SLP to SQP and back again is in CONOPT3 based on monitoring failure. The logic has been changed in CONOPT4 so transition is based on continuous measurements of curvature, both in the general constraints and in the objective function, combined with estimates of computational costs and progress for SLP and SQP.

The continuation of the log file for GAMS Library model otpop.gms shows some of this:

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Objective</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>2.3861517936E+02</td>
<td>3.6E+02</td>
<td>12</td>
<td>2.1E-01</td>
<td>7</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1.4308470370E+02</td>
<td>1.1E+02</td>
<td>16</td>
<td>2.0E-01</td>
<td>9</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>9.4375606705E+01</td>
<td>1.5E+02</td>
<td>16</td>
<td>1.8E-01</td>
<td>5</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2.4652509772E+01</td>
<td>7.4E+01</td>
<td>16</td>
<td>6.7E-01</td>
<td>2</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2.4445151316E-02</td>
<td>3.3E+01</td>
<td>16</td>
<td>1.0E+00</td>
<td>6</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>5.0735392100E-06</td>
<td>4.4E+00</td>
<td>16</td>
<td>1.0E+00</td>
<td>5</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>1.0276261632E-09</td>
<td>1.9E-02</td>
<td>16</td>
<td>1.0E+00</td>
<td>3</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>1.4326828955E-13</td>
<td>2.8E-06</td>
<td>16</td>
<td>1.0E+00</td>
<td>1</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>1.4326828955E-13</td>
<td>4.0E-10</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Optimal solution. Reduced gradient less than tolerance.

Iterations 6 to 9 are SLP iterations (Phase 3) and iterations 10 to 14 are SQP iterations (Phase 4). The SLP iterations have Step less than 1 due to nonlinear objective terms and CONOPT4 jumps directly from SLP to SQP. CONOPT3 needed some steepest descend and some Quasi-Newton iterations as a transition, and although these iterations are fast, the transition could sometimes be slow.

5.8.7 Bad Iterations

Bad iterations (for both solvers flagged with “F” in the “OK” column of the log file) are an important problem for both CONOPT3 and CONOPT4. They appear if the output from the SLP or SQP procedure is a search direction where CONOPT cannot move very far because it is difficult to make the nonlinear constraints feasible again. The efforts spend in solving the SLP or SQP procedure is therefore partially wasted. The problem is usually associated with a basis that is ill-conditioned or changes very fast.

CONOPT3 has some heuristics for changing the basis in these cases. For models with many basic and superbasic variables this can be a slow and not very reliable procedure.

CONOPT4 uses a more rigorous procedure based on monitoring the size of some intermediate terms. It can be the size of elements of the tangent for basic variables relative to similar elements for superbasic variables. Or it can be intermediate results from the computation of the reduced costs. The monitoring procedure is cheap and it is used to maintain a well-conditioned basis throughout the optimization instead of waiting until something does not work well.

5.8.8 Saddle Points and Directions of Negative Curvature

CONOPT is based on moving in a direction derived from the gradient (or the reduced gradient for models with constraints). If the reduced gradient (projected on the bounds) is zero then the solution satisfies the first-order optimality conditions and it is standard procedure to stop. Unfortunately, this means that we can stop in a saddle-point.

It is not very common to move towards a saddle-point and get stuck in it. However, it is not uncommon that the initial point, provided by a user or by default, is a saddle point. A simple example is the
constraint $x \cdot y = 1$ started with $x.l = y.l = 0$ that easily can end with a locally infeasible solution. Or minimize $z$, $z = x \cdot y$ with the same starting point that could end locally optimal without moving even though better points exist in the neighborhood.

CONOPT4 has an added procedure that tries to find a direction of negative curvature that can move the solution point away from a saddle-point. The procedure is only called in points that satisfy the first order optimality conditions and it is therefore a cheap safeguard. The theory behind the method is developed for models without degeneracy and it works very well in practice for these models. Models with some kind of degeneracy (basic variables at bound or nonbasic variables with zero reduced cost) use the same procedure, but it is in this case only a heuristic that cannot be guaranteed to find a direction of negative curvature, even if one exists.

If you know that there are no directions of negative curvature you can turn the procedure of by setting the logical option Flg_NegCurve to false. If the model is known to be convex you can set the logical option Flg_Convex to true and it will also turn this procedure off. The saving is usually very small, except for models that solve in very few iterations and for model with a large number of superbasics.

There is no output in the log file for negative curvature. If a useful direction is found CONOPT4 will follow it and the optimization continues. Otherwise, the solution is declared locally optimal.

5.8.9 Use of Alternative Sub-Models

During the course of an optimization CONOPT4 can work with up to three different internal sub-models. These models are:

- Full Model: This model consists of the constraints in the user's model excluding all pre- and post-triangular constraints and with the definitional variables eliminated by their defining constraints.
- No-Penalty Model: This model consists of the Full Model excluding all penalty and mini-max constraints. This model does not have an objective function.
- Linear Feasibility Model: This model consists of the linear constraints of the Full Model. The Linear Feasibility model is either solved without an objective function or minimizing a quadratic distance measure; this is discussed below.

The pre-triangular variables are considered fixed and they do not appear in any of the sub-models. Their influence comes through their contribution to coefficients and constant terms. The post-triangular variables are considered intermediate variables in the definition of the objective function. They do not appear in the last two models that only are concerned with feasibility, and they only appear indirectly via the objective in the Full Model. The defined variables are considered intermediate variables in the definition of the remaining constraints in the same way as post-triangular variables are intermediate in the objective. The variables in the Full Model are all variables excluding pre- and post-triangular variables and excluding defined variables; this set can include variables that do not appear in any constraints. The constraints of the full models are all constraints excluding pre- and post-triangular constraints and with the definitional equations logically eliminated. The variables in the Linear Feasibility Model and in the No-Penalty Model are the variables that appear in the constraints of these models (excluding pre-triangular variables).

CONOPT always starts by searching for a feasible solution and the sub-models only play a role in this part of the optimization so if the initial point provided by the modeler is feasible then these sub-models are irrelevant. If there are many penalty and/or minimax constraints then the No-Penalty Model will be much smaller than the Full Model and it is more efficient to use the smaller model while searching for feasibility. So the No-Penalty model is only introduced for efficiency reasons. It is by default solved before the Full Model if all of the following conditions are satisfied:

- The Flg_NoPen options is true (the default value)
- The model is not a CNS model
- The user did not provide an initial basis
- Some of the constraints in the No-Penalty Model are infeasible.
- The number of penalty and minimax constraints is more than the number of constraints in the Full Model multiplied by the value of option Rat.NoPen. The default value of Rat.NoPen is 0.1, i.e. the No-Penalty Model is only defined and solved if it is at least 10% smaller than the Full Model.

The GAMS Library model prolog.gms used earlier has many penalty and minimax constraints. The relevant part of the log file is:

```
Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
0 0 1.2977431398E+03 (Input point)
The post-triangular part of the model has 1 constraints and variables.
There are 17 penalty and minimax constraints with 3 variables.
Reduced model without penalty components has 17 variables and 5 constraints with 46 Jacobian elements, 0 of which are nonlinear.

Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
1.2960000000E+03 (Model without penalty constraints)
5.0625000000E+00 (After scaling)
0.0000000000E+00 (After adjusting individual variables)
```

Previous model terminated and penalty components are added back in.
Full preprocessed model has 20 variables and 22 constraints with 120 Jacobian elements, 8 of which are nonlinear.

```
Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
0.0000000000E+00 (After adjusting penalty variables)
```

** Feasible solution. Value of objective = -1147.54681404

The No-Penalty model is small with only 5 constraints and 17 variables and it solves already in the 'Adjust Individual Point' procedure. The Full Model is significantly larger with 22 constraints and 20 variables, and it is feasible when started from the solution to the No-Penalty model.

The Linear Feasibility Model is introduced to help avoid locally infeasible solutions. It produces a starting point to the nonlinear models (No-Penalty Model or Full Model) that satisfies all linear constraints. If the Linear Feasibility Model is infeasible then the overall model is proved to be infeasible (independent of nonlinearities) and there is no reason to proceed with the nonlinear part of the model.

The Linear Feasibility Model is only useful if the model has some linear constraints and if the initial point provided by the modeler does not satisfy these constraints. If a feasible solution to the linear constraints is found there are several possible ways to continue before the No-Penalty Model and/or the Full Model are started:

A. Use the solution point as is.

B. Perform an approximate minimization of the weighted distance from the user's initial point. Include only the variables that have non-default initial values, i.e. variables with an initial value (xini) that is different from zero projected on the bounds, i.e. xini ne min(max(0,x.lo),x.up). The distance measure is \(\text{sqr}\left(\frac{x-xini}{\max(1,\text{abs}(xini))}\right)\).

C. As in B, but include all variables in the distance measure.

D. As in C, but define xini to 1 projected on the bounds for all variables with default initial value.
Possibility A is fast but may give a starting point for the nonlinear model far from the initial point provided by the user, B is slower but gives a starting point for the nonlinear model that is close to the point provided by the user, and C and D are also slower but may provide reasonably good and different starting points for the nonlinear model.

The order in which the sub-models are solved depends on a Linear Feasibility Model strategy option, \texttt{Lin\_Method}:

1. If \texttt{Lin\_Method} has the default value 1 then the initial point and basis is assumed to be fairly good and CONOPT4 will start with the No-Penalty Model (only if the conditions mentioned above are satisfied) followed by the Full Model. If the model terminates locally optimal, unbounded, or on some resource limit (time, iterations, function evaluations) then we are done and CONOPT terminates. But if the model is locally infeasible then we build and solve the Linear Feasibility Model. If this model is infeasible, the overall model is infeasible and we are again done. If it is feasible we minimize objective B and use the solution point as a second starting point for the nonlinear model. If this attempt also terminates locally infeasible we try to generate an alternative initial point with objective C and then with objective D. If all fails, the model is labeled locally infeasible.

2. With \texttt{Lin\_Method} = 2 CONOPT will start with the Linear Feasibility Model with objective A before looking at the No-Penalty and Full models. If they are locally infeasible from this starting point we followed the procedure from above with objective B, C, and then D.

3. \texttt{Lin\_Method} = 3 is similar to \texttt{Lin\_Method} = 2 except that the first objective A is skipped.

An example where all Linear Feasibility objectives are used is taken from \texttt{ex5.3.2.gms} in the GlobalLib collection of test models. It shows the repeated starts of the Linear Feasibility Model followed by the Full Preprocessed model:

Preprocessed model has 22 variables and 16 constraints with 59 Jacobian elements, 24 of which are nonlinear.

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1200000000E+02</td>
<td>1</td>
<td>1</td>
<td>6.2500000000E-01</td>
<td>0.0E+00</td>
<td>5</td>
<td>0.0E+00</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

** Infeasible solution. Reduced gradient less than tolerance.

Initial linear feasibility model has 22 variables and 7 constraints with 22 linear Jacobian elements.

Objective: Distance to initial point (nondefault variables)

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0200000000E+02</td>
<td>1</td>
<td>1</td>
<td>6.2500000000E-01</td>
<td>0.0E+00</td>
<td>5</td>
<td>0.0E+00</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

** Linear constraints feasible. Distance = 0.0000000000

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Distance</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>0.0000000000E+00</td>
<td>0.0E+00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
** Infeasible solution. There are no superbasic variables.

Restarting linear feasibility model.  
Objective: Distance to initial point (all variables)

Iter Phase  Ninf  Infeasibility  RGmax  NSB  Step  InItr  MX  OK
0.0000000000E+00 (Linear feasibility model)  
0.0000000000E+00 (After scaling)

** Linear constraints feasible. Distance = 90002.0000000

Iter Phase  Ninf  Distance  RGmax  NSB  Step  InItr  MX  OK
16 4 2.2501000000E+04 1.8E-12 5

Restarting preprocessed model from a new starting point.

Iter Phase  Ninf  Infeasibility  RGmax  NSB  Step  InItr  MX  OK
1.8492500000E+02 (Full preprocessed model)  
1.0775781250E+01 (After scaling)

21 1 1 6.2500000000E-01 5.4E-03 3 0.0E+00 T T
26 1 1 6.249999982E-01 3.7E-07 3 2.4E+03 T T
27 2 1 6.2500000000E-01 0.0E+00 2

** Infeasible solution. Reduced gradient less than tolerance.

Restarting linear feasibility model.  
Objective: Distance to point away from bounds

Iter Phase  Ninf  Infeasibility  RGmax  NSB  Step  InItr  MX  OK
1.4210854715E-14 (Linear feasibility model)  
5.5511151231E-17 (After scaling)

** Linear constraints feasible. Distance = 15.1922028230

Iter Phase  Ninf  Distance  RGmax  NSB  Step  InItr  MX  OK
31 4 2.5570990132E+00 1.1E-04 13 1.0E+00 2 F T
32 4 2.5570990132E+00 7.7E-11 13

Restarting preprocessed model from a new starting point.

Iter Phase  Ninf  Infeasibility  RGmax  NSB  Step  InItr  MX  OK
6.0836352222E+02 (Full preprocessed model)  
9.7890253865E+00 (After scaling)

35 S0 0 2.3859378417E+00 1.0E+00 3 T T

** Feasible solution. Value of objective = 1.86415945946

Iter Phase  Ninf  Objective  RGmax  NSB  Step  InItr  MX  OK
40 4 1.8641594595E+00 0.0E+00 0

** Optimal solution. There are no superbasic variables.
The Full Model is infeasible when started directly from the values provided by the user. The Linear Feasibility model is then solved to get a different starting point for the Full model where the linear part is feasible. The Full Model is also infeasible from this point and it is necessary to solve the Linear Feasibility model three times with different objective functions before the full model becomes feasible.

The number of submodels that are solved is limited by the option, Num_Rounds. The default value is 4, i.e. we will try very hard to find a feasible point as shown in the example above. The value 1 will make CONOPT terminate immediately when a locally infeasible point is found. You can use Num_Rounds if you are not interested in spending a lot of time on a model that is likely to be infeasible. It can be particularly relevant if CONOPT4 is used as the sub-solver inside SBB where infeasible sub-problems are fairly likely.

If the model is defined to be convex with the option Flg_Convex = true then a locally infeasible solution is labeled (globally) infeasible and the Linear Feasibility Model will not be used. (A locally optimal solution is also labeled (globally) optimal.)

5.8.10 Multiple Threads

CONOPT4 can use multiple threads for some internal computations and in collaboration with GAMS for function and derivative evaluations. Multiple threads are currently only used for certain fairly large and dense computations and there are not so many of these in the types of model usually build with GAMS. In addition, multiple threads have quite high overhead and they are therefore only useful for fairly large models. Currently the best improvements have been for very large models with more than 100,000 variables or constraints, in particular for CNS models if this size.

It is the intention to implement multiple threads into more parts of CONOPT4 in the future.

Threads can be turned on with the GAMS command-line option Threads=n or with the CONOPT4 option threads.

5.8.11 Solving CNS Models

CONOPT4 is a general NLP (Nonlinear Programming) solver, but it also has some specialized routines for solving CNS (Constrained Nonlinear System) models. Due to the specialized routines, it is in general possible to solve much larger CNS models than NLP models, and there are users that solve models with more than 1 million variables and equations, mainly in the area of Economic General Equilibrium models.

5.8.11.1 Basic CNS Theory

CONOPT4’s CNS procedure is based on Newton’s method: The constraints are linearized around the starting point and the suggested change in variables is computed by solving a square set of linear equations where the right-hand-side is the vector of residuals:

\[ g(x_0 + dx) = 0 \]  

is approximated by

\[ g(x_0) + J \cdot dx = 0 \iff dx = J^{-1} g(x_0) \]  

The matrix \( J \) is the Jacobian or the matrix of first derivatives of the constraint functions \( g \). For a CNS model \( J \) will be a square matrix. In the optimization part of CONOPT4 we work with a sub-matrix selected from a subset of columns of the Jacobian and this matrix is called a Basis and we will therefore sometimes use the term basis and inverse basis when referring to \( J \) and \( J^{-1} \). If the starting point is close to the solution, Newton’s method is known to converge very quickly. We may even use the same \( J^{-1} \) for several iterations. The rate of convergence, defined as the norm of \( g \) after the iteration relative to the
norm of $g$ before the iteration, using a fixed $J^{-1}$ is approximately equal to the largest eigenvalue in the matrix

$$I - J_I^{-1} J_F$$  \hspace{1cm} (5.3)

where $J_I$ is the initial Jacobian matrix, that is used in the inversion, and $J_F$ is the final Jacobian matrix, computed the final solution point. Approximating $J_F = J_I + dJ$ we get an approximation of the rate of convergence, $\rho$, as

$$\rho = \max_{\text{eigenvalue}} \left( I - J_I^{-1} (J_I + dJ) \right) = \max_{\text{eigenvalue}} \left( J_I^{-1} dJ \right)$$  \hspace{1cm} (5.4)

When the change in Jacobian, $dJ$, is small, the largest eigenvalue will also be small and we will get fast convergence. If the starting point is far from the final point, then the linearization may not be very good with a large $dJ$ and the suggested change in variables may give rise to growing residuals. The procedure used inside CONOPT4’s CNS solver is to find a step in the suggested direction, $dx$, where the residuals are approximately minimized and use this step. The solution will gradually come closer, the linearization will become better, $dJ$ will become smaller, the optimal step will come closer to one, and the procedure will (usually) terminate with fast convergence.

5.8.11.2 Evaluating External Chocks using CNS

Economic General Equilibrium / CNS models are often used to evaluate the effect of policies changes or external chocks. The setup in GAMS is often of this type:

* Define model and initialize data
  Solve M using CNS;

* Store the base solution

* Define the chock by changing some exogenous variables
  Solve M using CNS;

* Compare the new solution with base solution.

The second `Solve` will start from the base solution and this point will in many cases be a very good starting point and the second `Solve` will therefore run quickly. However, for large chocks and very complex models, the second `Solve` may converge very slowly and it may not solve at all, because the new solution is too far from the base solution. For these cases it is not unusual to break the large chock down into a sequence of smaller chocks and solve the model several times. With smaller chocks each `Solve` will start from a point that is not too far away from the final solution for this chock. The GAMS model with smaller chocks will look like this:

* Define model and initialize data
  Solve M using CNS;

* Store the base solution

        Parameter ExoDiff(indx) The change in exogenous variables;
        ExoDiff(indx) = ExoTarget(indx) - ExoBase(Indx);

        Set Iter / Iter1*Iter5 /
        Parameter Step(Iter); Step(Iter) = ord(Iter)/card(Iter);
        Loop(Iter,
          Exo(indx) = ExoBase(indx) + Step(Iter)*ExoDiff(indx);
          Solve M using CNS;
        );

* Compare new solution with base solution.

The number of iterations and the way the `Step`-parameter is defined may vary and there is often no good a-priori rule for deciding. The smaller chocks will give $dJ$-matrices with smaller value and therefore smaller (equal to better) rates of convergence. Easy experiments can be solved with few steps while difficult experiments require more steps to converge. More iterations will give smaller steps and the individual `Solve` will be faster, but the overhead in model generation and extra solves can be large. For some chocks there is no solution to the model: some of the model variables may go outside their bounds, some may move towards infinity, or the model may no longer be well-defined because the basis matrix becomes singular. In practice, one of the `Solve` in the `Loop` will violate a bound or it will converge extremely slowly, but it is often not possible to pinpoint exactly how large the chock should be before the model breaks down.
5.8.11.3 Evaluating External Chocks using NLP

The theoretical solution to avoiding the choice of iteration steps is to change the model to an NLP and maximize the chock:

\* Define model and initialize data
Solve M using CNS;
\* Store the base solution
Parameter ExoDiff(indx) The change in exogenous variables
ExoDiff(indx) = ExoTarget(indx) - ExoBase(Indx);
\* Define the chock using variables and equations
Variable Exov(indx) Exogenous variables as a function of step;
Variable Step Share of the chock;
Equation Exod(indx) Define Exov;
Exod(indx).. Exov(indx) =E= ExoBase(indx)+Step*ExoDiff(indx);
Model m1 / M + Exod /
Step.lo = 0; Step.up = 1; Step.l = 0;
Exov.l(indx) = ExoBase(indx) + Step.l*ExoDiff(indx);
Solve M1 using NLP maximizing Step;
\* Compare new solution with base solution.

The NLP approach follows the solution from the feasible solution at Step = 0 through solutions for larger and larger steps and the change in Step is determined by how easy it is to maintain feasibility of the constraints. This NLP works well for small or simple models. However, for very large models such as the 1 million plus models mention above, the overhead associated with NLP can be very large because the specialized CNS routines no longer are used. Some statistics using CONOPT 4.26 on a 6 million model shows the problems:

- Pre-processor: The special CNS version locates pre- and post-triangular variables in 5 seconds; the NLP version used another 25 second to derive bounds and find defined variables. Analysing the 2 mill defined variables took 4000 seconds; it is now reduced to 8 seconds in CONOPT 4.27.
- Initial Basis: The special CNS version assigns all non-fixed variables to the basis and this is very fast; the NLP version makes sure that the basis is non-singular and it takes 1000 seconds, and the basis is not complete and many hours are needed to repair it.

5.8.11.4 Option TraceCNS

The NLP model defined above is not a general NLP. It has exactly one degree of freedom, namely the Step variable (the objective variable in the model), and the rest of the model, including the extra Exod-equations and Exov variables, is a CNS model. To take advantage of the specialized CNS routines CONOPT 4.27 introduced a new logical option, Flg_TraceCNS or just TraceCNS. If Flg_TraceCNS is defined as true (default is false), then CONOPT will use the special CNS pre-processor, it will define the initial CNS basis, it will fix the objective variable (here Step) to its initial value and find an initial feasible solution using the old CNS procedure. When a feasible solution has been found, the bounds on the objective variable are reset and the standard optimization routine is started from a feasible solution and a good basis. There are a few adjustments in the optimization routine; some are related to the iteration output, and some take advantage of the special properties of the TraceCNS model.

An example of the log-file for a large model can explain some of the key aspects. The chock is a tax-rate that is changed from a base value of 500 to the target of 1250:
Will use up to 12 threads.

15674 constant coefficients less than 1.0E-20 (=Tol_Zero) have been removed. 20865735 coefficients left.

The user model has 6444864 constraints and 6444865 variables with 20865735 Jacobian elements, 8039142 of which are nonlinear.

<table>
<thead>
<tr>
<th>Iter Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5.8480729607E+07</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The pre-triangular part of the model has 2547274 constraints and variables. The post-triangular part of the model has 1174195 constraints and variables.

Preprocessed model has 2723395 constraints and 2723396 variables with 10037529 Jacobian elements, 5065730 of which are nonlinear.

<table>
<thead>
<tr>
<th>Iter Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5.2563247343E+04</td>
<td>6.7E-02</td>
<td>1</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.8116053702E+01</td>
<td>1.0E+00</td>
<td>7</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3.5896510562E-10</td>
<td>1.0E+00</td>
<td>13</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Feasible solution to a square system.

** Starting TraceCNS from Objective = 500.000000066

<table>
<thead>
<tr>
<th>Iter Phase</th>
<th>Ninf</th>
<th>Objective</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>6.87500000005E+02</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8.5383670354E+02</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0488459082E+03</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1.0488459082E+03</td>
<td>2.6E+02</td>
<td>1</td>
<td>7.3E-01</td>
<td>1</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.2500000000E+03</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** TraceCNS terminates with objective at optimal bound.

The original CNS model had 6285822 equations. We have added 159042 equations to define the 159042 exogenous variables, that have been made endogenous (representing the Exod equations and Exov variables above). Since these extra equations have a very simple structure, they will not add much complexity to the solution process.

In this case we do not start from a feasible solution. The first iterations down to the line Feasible solution to a square system are CNS iterations. Each line represents an inversion of the basis and the numbers in the InItr column are the number of Newton iterations that use the same basis.

Starting TraceCNS .. identifies the start of the NLP procedure and the Objective defines the starting step. The numbers in the Objective column represent the progress of the step or here the tax rate. There are two types of iteration lines. The lines without a number in the Iter column represents a re-computation of the Jacobian and inversion of the basis inside the linesearch and the numbers in the InItr column are the number of Newton iterations using this basis. The lines with a number in the Iter column are the standard output from CONOPT4's NLP procedure and the number in the InItr column
are the number of LP or QP iterations used to define the search direction, usually just 1. The last line without a value in the Iter column will have the same Objective value as the line with a value in the Iter column. The new iteration lines with a number in the Iter column have been added to show that the solution process is alive and the InItr numbers show how much work is done. In very large CNS models each inversion may take more than one minute, and one linesearch or NLP iteration with several inversion may take much longer. Because progress may be slow, CONOPT4 will check for time or resource limit and for user interrupt (Cntrl-C) after each of these lines.

Inside the linesearch CONOPT4 will use Newton iterations to restore feasibility as discussed in Basic CNS Theory. The change in Jacobian, $dJ$, will approximately grow with the size of the step, $dJ = M \cdot Step$, where $M$ is a matrix of 2nd derivatives in the direction we are moving. The rate of convergence can therefore be approximated by

$$\rho(Step) = \max_{\text{eigenvalue}} (J^{-1}M \cdot Step) = \max_{\text{eigenvalue}} (J^{-1}M) \cdot Step = \rho(1) \cdot Step$$  \hspace{1cm} (5.5)

CONOPT4 will approximate $\rho(1)$ and use it to decide on the largest Step that will lead to convergence. Once we have moved this step the Jacobian has changed so much that further progress is impossible with the current Jacobian and its inverse, and it is necessary to recompute the Jacobian and its inverse. Since this process is the most expensive part of the optimization procedure, often accounting for more than 75% of the overall time for large models, we use the steps where the Jacobian is recomputed and reinverted to identify the overall progress.

The final line **TraceCNS terminates with objective at the optimal bound** tells us that the tax rate has been changed to the target value and that the model is solved with this value. Since the model is considered a special case of a CNS model the solution returned to GAMS will not have any dual variables, just as for regular CNS models.

The particular case was originally solved using four CNS Solve statements. The total time was 12.43 minutes with 5.23 minutes in the four CONOPT4 runs and 7.20 minutes in GAMS where most of the time was used for the four model generations.

With the NLP model and the Flg_TraceCNS option the total time was 6.47 minutes; CONOPT4 used 5.20 minutes and since there is only one model generation, GAMS only used 1.27 minutes. Notice that the time spend in CONOPT4 is about the same, but there is a big saving in the time spend in GAMS. It should be mentioned that the same NLP model without the Flg_TraceCNS option ran for 36 hours without reaching a solution.

5.8.11.5 Option Trace_MinStep

A more difficult case was also originally solved using four CNS Solve statements. The two first CNS models solved in 2 min 23 seconds and 1 min 31 seconds, respectively. The third CNS had problems; it used 1 hour 5 minutes and the last part of the iteration log was:

<table>
<thead>
<tr>
<th>Iter</th>
<th>Phase</th>
<th>Ninf</th>
<th>Infeasibility</th>
<th>RGmax</th>
<th>NSB</th>
<th>Step</th>
<th>InItr</th>
<th>MX</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>0</td>
<td>0</td>
<td>5.2658286648E+02</td>
<td></td>
<td></td>
<td>1.3E-02</td>
<td>1 F T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>0</td>
<td>0</td>
<td>5.212264122E+02</td>
<td></td>
<td></td>
<td>1.2E-02</td>
<td>1 F T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>0</td>
<td>0</td>
<td>5.1523124045E+02</td>
<td></td>
<td></td>
<td>1.2E-02</td>
<td>1 F T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>0</td>
<td>0</td>
<td>5.0937544127E+02</td>
<td></td>
<td></td>
<td>1.2E-02</td>
<td>1 F T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>0</td>
<td>0</td>
<td>5.0359613057E+02</td>
<td></td>
<td></td>
<td>1.0E-02</td>
<td>1 F T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Domain error(s) in nonlinear functions.**
Check bounds on variables.

The long time is caused by the many small steps, and the solution process finally terminates because a variable, $x$, that appears in an $x^{**p}$ expression with $p < 1$ becomes negative. The same model solved as an NLP model with Flg_TraceCNS solved in 28 min 10 sec. The first part of the iteration log is similar to the one shown above. The last part of the iteration log is:
** TraceCNS. Convergence too slow. Change in objective between reinversion has been less than 0.75 for three consecutive reinversions. 

The objective or the step changes more and more slowly between the reinversions. This indicates that the Jacobian changes faster and faster so the change, $dJ$, is about the same size as the initial Jacobian after smaller and smaller steps.

Unfortunately, it is not easy to identify where and why the Jacobian changes too much. It requires knowledge of the actual model, and an expert user must combine this knowledge with the final solution reported in GAMS. The problem is often some variables that become very large or very small.

By default, CONOPT4 will stop when the change in objective between reinversions is less than 0.1% of the target step for three consecutive reinversions. This tolerance can be changed with option Trace_MinStep to get a more or less accurate solution. A smaller value will seldom change much but it can be costly in time.

$\text{Trace}_{\text{MinStep}} = 0$ forces CONOPT4 to move as far as possible. For this model the final step changes from 653.2 to 657.9 compared to the start at 500 and the target at 1250, and the solution time changes from 28 minutes to 3 hours 20 minutes.

### 5.8.11.6 Termination Messages from TraceCNS

The first example showed the ideal termination message

** TraceCNS terminates with objective at optimal bound. 

Sometimes the objective cannot be moved all the way to the target value. There are several different cases:
• We saw the slow convergence message in the previous section.

** CONOPT may terminate with messages like:**

- **TraceCNS: An endogenous variable has reached its bound.**

The solution process is stopped because a variable has reached a bound. The offending bound will be shown in the listing file with a message like:

**** ERRORS/WARNINGS IN VARIABLE x(i2)

1 error(s): The variable has reached its bound

In this case the solution will include dual variables.

• Another termination message is:

** TraceCNS: The basis has become singular.**

The basis has become singular at the final step and it is not possible to move further. Unfortunately, it is not possible to give a detailed description of a variable or equation that causes the problem; detailed knowledge of the actual model must be combined with the solution. The solution returned to GAMS will include dual variables that may help in the analysis. Most models that could terminate with a singular basis will in practice terminate with a slow convergence message because the path to the final singular basis often will require a large number of small steps.

• CONOPT4 may also terminate due to functions evaluation error with a message like this:

** Domain error(s) in nonlinear functions.**

Check bounds on variables.

The equation(s) with function evaluation errors will be shown in the listing file with a message like:

**** ERRORS/WARNINGS IN EQUATION e(i1)

2 error(s): sqrt: FUNC DOMAIN: x < 0 (RETURNED 0)

• Finally, CONOPT4 may terminate due to an unbounded variable with a message like this:

** Unbounded solution. A variable has reached 'Infinity'.**

Largest legal value (Lim_Variable) is 1.00E+15

The variable that has become too large will be shown in the listing file with a message like:

**** ERRORS/WARNINGS IN VARIABLE x(i3)

1 error(s): The variable has reached 'infinity'
5.8.11.7 Error Messages from TraceCNS

**Flag TraceCNS** can only be used on a very special type of model and if the requirements are not satisfied you may see some of the following messages.

First of all, the model must be an optimization model with an objective variable. This is tested by GAMS during compile time and an error message will come from GAMS.

**Flag TraceCNS** requires that the objective variable is bounded in the optimization direction. If this is not satisfied you will get this message:

** Error in TraceCNS: The objective must be bounded.

after the pre-processor.

There must only be one degree of freedom represented by the objective variable. If there are extra free variables you will get this message:

** Error in TraceCNS: The number of non-fixed variables excluding the objective (nn) is not equal to the number of equations (mm).

also right after the pre-processor.

5.8.12 APPENDIX A - Options

The options that ordinary GAMS users can access are listed below.

5.8.12.1 Algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flg_Conv</td>
<td>Flag for defining a model to be convex</td>
<td>0</td>
</tr>
<tr>
<td>Flg_Crash</td>
<td>Flag for pre-selecting slacks for the initial basis.</td>
<td>0</td>
</tr>
<tr>
<td>Flg_Dbgs</td>
<td>Flag for debugging interval evaluations.</td>
<td>0</td>
</tr>
<tr>
<td>Flg_NegCrv</td>
<td>Flag for testing for negative curvature when apparently optimal</td>
<td>1</td>
</tr>
<tr>
<td>Flg_NoPen</td>
<td>Flag for allowing the Model without penalty constraints</td>
<td>1</td>
</tr>
<tr>
<td>Flg_SLP</td>
<td>Flag for enabling SLP mode.</td>
<td>1</td>
</tr>
<tr>
<td>Flg_SQP</td>
<td>Flag for enabling of SQP mode.</td>
<td>1</td>
</tr>
<tr>
<td>Flg_Sqr</td>
<td>Flag for Square System. Alternative to defining modeltype=CNS in GAMS</td>
<td>0</td>
</tr>
<tr>
<td>Flg_Trace</td>
<td>Flag for tracing a CNS solution.</td>
<td>0</td>
</tr>
<tr>
<td>Frq_Resp</td>
<td>Rescaling frequency.</td>
<td>5</td>
</tr>
<tr>
<td>Lim_Err</td>
<td>Limit on errors in Directional Second Derivative evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>Lim_Err</td>
<td>Limit on number of function evaluation errors. Overwrites GAMS Domlim option</td>
<td>GAMS DomLim</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lim_Err_Hessian</td>
<td>Limit on errors in Hessian evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>Lim_Iteration</td>
<td>Maximum number of iterations. Overwrites GAMS Iterlim option.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>Lim_NewSuper</td>
<td>Maximum number of new superbasic variables added in one iteration.</td>
<td>auto</td>
</tr>
<tr>
<td>Lim_RedHess</td>
<td>Maximum number of superbasic variables in the approximation to the Reduced Hessian.</td>
<td>auto</td>
</tr>
<tr>
<td>Lim_SlowPrg</td>
<td>Limit on number of iterations with slow progress (relative less than Tol_Obj Change).</td>
<td>20</td>
</tr>
<tr>
<td>Lim_StallIter</td>
<td>Limit on the number of stalled iterations.</td>
<td>100</td>
</tr>
<tr>
<td>Lim_Start_Degen</td>
<td>Limit on number of degenerate iterations before starting degeneracy breaking strategy.</td>
<td>100</td>
</tr>
<tr>
<td>Lim_Time</td>
<td>Time Limit. Overwrites the GAMS Reslim option.</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>Lim_Variable</td>
<td>Upper bound on solution values and equation activity levels</td>
<td>1.e15</td>
</tr>
<tr>
<td>Lin_Method</td>
<td>Method used to determine if and/or which Linear Feasibility Models to use</td>
<td>1</td>
</tr>
<tr>
<td>Mtd_Dbgs1Drv</td>
<td>Method used by the function and derivative debugger.</td>
<td>0</td>
</tr>
<tr>
<td>Mtd_RedHess</td>
<td>Method for initializing the diagonal of the approximate Reduced Hessian</td>
<td>0</td>
</tr>
<tr>
<td>Mtd_Scale</td>
<td>Method used for scaling.</td>
<td>3</td>
</tr>
<tr>
<td>Mtd_Step_Phase0</td>
<td>Method used to determine the step in Phase 0.</td>
<td>auto</td>
</tr>
<tr>
<td>Mtd_Step_Tight</td>
<td>Method used to determine the maximum step while tightening tolerances.</td>
<td>0</td>
</tr>
<tr>
<td>Num_Rounds</td>
<td>Number of rounds with Linear Feasibility Model</td>
<td>4</td>
</tr>
<tr>
<td>Rat_NoPen</td>
<td>Limit on ratio of penalty constraints for the No Penalty model to be solved</td>
<td>0.1</td>
</tr>
<tr>
<td>Tol_Bound</td>
<td>Bound filter tolerance for solution values close to a bound.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>Tol_BoxSize</td>
<td>Initial box size for trust region models for overall model</td>
<td>10</td>
</tr>
<tr>
<td>Tol_BoxSize_Lin</td>
<td>Initial box size for trust region models for linear feasibility model</td>
<td>1000</td>
</tr>
<tr>
<td>Tol_Box_LinFac</td>
<td>Box size factor for linear variables applied to trust region box size</td>
<td>10</td>
</tr>
<tr>
<td>Tol_Def_Mult</td>
<td>Largest growth factor allowed in the block of definitional constraints</td>
<td>1.e4</td>
</tr>
<tr>
<td>Tol_Feas_Max</td>
<td>Maximum feasibility tolerance (after scaling).</td>
<td>1.e-7</td>
</tr>
<tr>
<td>Tol_Feas_Min</td>
<td>Minimum feasibility tolerance (after scaling).</td>
<td>4.e-10</td>
</tr>
<tr>
<td>Tol_Feas_Tria</td>
<td>Feasibility tolerance for triangular equations.</td>
<td>1.0.e-8</td>
</tr>
<tr>
<td>Tol_Fixed</td>
<td>Tolerance for defining variables as fixed based on initial or derived bounds.</td>
<td>4.e-10</td>
</tr>
<tr>
<td>Tol_Jac_Min</td>
<td>Filter for small Jacobian elements to be ignored during scaling.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>Tol.Linesearch</td>
<td>Accuracy of One-dimensional search.</td>
<td>0.2</td>
</tr>
<tr>
<td>Tol_Obj_Acc</td>
<td>Relative accuracy of the objective function.</td>
<td>3.0e-13</td>
</tr>
<tr>
<td>Tol_Obj_Change</td>
<td>Limit for relative change in objective for well-behaved iterations.</td>
<td>3.0e-12</td>
</tr>
<tr>
<td>Tol_Optimality</td>
<td>Optimality tolerance for reduced gradient when feasible.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Tol_Opt_Infeas</td>
<td>Optimality tolerance for reduced gradient when infeasible.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>Tol_Opt_LinF</td>
<td>Optimality tolerance when infeasible in Linear Feasibility Model</td>
<td>1.e-10</td>
</tr>
<tr>
<td>Tol_Piv_Abs</td>
<td>Absolute pivot tolerance.</td>
<td>1.e-10</td>
</tr>
<tr>
<td>Tol_Piv_Abs_Ini</td>
<td>Absolute Pivot Tolerance for building initial basis.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>Tol_Piv_Abs_NLTr</td>
<td>Absolute pivot tolerance for nonlinear elements in pre-triangular equations.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>Tol_Piv_Ratio</td>
<td>Relative pivot tolerance during ratio-test</td>
<td>1.e-8</td>
</tr>
<tr>
<td>Tol_Piv_Rel</td>
<td>Relative pivot tolerance during basis factorizations.</td>
<td>0.05</td>
</tr>
<tr>
<td>Tol_Piv_Rel_Ini</td>
<td>Relative Pivot Tolerance for building initial basis</td>
<td>1.e-3</td>
</tr>
<tr>
<td>Tol_Piv_Rel_Updt</td>
<td>Relative pivot tolerance during basis updates.</td>
<td>0.05</td>
</tr>
<tr>
<td>Tol_Scale2D_Min</td>
<td>Lower bound for scale factors based on large 2nd derivatives.</td>
<td>1.e-6</td>
</tr>
<tr>
<td>Tol_Scale_Max</td>
<td>Upper bound on scale factors.</td>
<td>1.e25</td>
</tr>
<tr>
<td>Tol_Scale_Min</td>
<td>Lower bound for scale factors computed from values and 1st derivatives.</td>
<td>1</td>
</tr>
<tr>
<td>Tol_Scale_Var</td>
<td>Lower bound on x in x*Jac used when scaling.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>Tol_Zero</td>
<td>Zero filter for Jacobian elements and inversion results.</td>
<td>1.e-20</td>
</tr>
<tr>
<td>Trace_MinStep</td>
<td>Minimum step between Reinversions when using TraceCNS.</td>
<td>0.001</td>
</tr>
</tbody>
</table>

### 5.8.12.2 Debugging options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flg_Interv</td>
<td>Flag for using intervals in the Preprocessor</td>
<td>1</td>
</tr>
<tr>
<td>Flg_Prep</td>
<td>Flag for using the Preprocessor</td>
<td>1</td>
</tr>
<tr>
<td>Flg_Range</td>
<td>Flag for identifying sets of ranged constraints</td>
<td>1</td>
</tr>
<tr>
<td>Lim_Dbq_1Drv</td>
<td>Flag for debugging of first derivatives</td>
<td>0</td>
</tr>
<tr>
<td>Lim_Hess_Est</td>
<td>Upper bound on second order terms.</td>
<td>1.e4</td>
</tr>
<tr>
<td>Lim_Msg_Dbq_1Drv</td>
<td>Limit on number of error messages from function and derivative debugger.</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.8.12.3 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frq_Log_Simple</td>
<td>Frequency for log-lines for non-SLP/SQP iterations.</td>
<td>auto</td>
</tr>
<tr>
<td>Frq_Log_SlpSqp</td>
<td>Frequency for log-lines for SLP or SQP iterations.</td>
<td>auto</td>
</tr>
<tr>
<td>Lim_Msg_Large</td>
<td>Limit on number of error messages related to large function value and Jacobian elements.</td>
<td>10</td>
</tr>
<tr>
<td>Lim_Pre_Msg</td>
<td>Limit on number of error messages related to infeasible pre-triangle.</td>
<td>25</td>
</tr>
</tbody>
</table>
5.8.12.4 Interface options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooptfile</td>
<td>Flag for computing and using directional 2nd derivatives.</td>
<td>auto</td>
</tr>
<tr>
<td>Flg_2DDir</td>
<td>Flag for computing and using 2nd derivatives as Hessian of Lagrangian.</td>
<td>auto</td>
</tr>
<tr>
<td>HEAPLIMIT</td>
<td>Maximum Heap size in MB allowed</td>
<td>1e20</td>
</tr>
<tr>
<td>HessianMemFac</td>
<td>Memory factor for Hessian generation: Skip if Hessian elements &gt; Nonlinear Jacobian elements*HessianMemFac, 0 means unlimited.</td>
<td>0</td>
</tr>
<tr>
<td>THREAD2D</td>
<td>Number of threads used for second derivatives</td>
<td>1</td>
</tr>
<tr>
<td>THREADC</td>
<td>Number of compatibility threads used for comparing different values of THREADS</td>
<td>1</td>
</tr>
<tr>
<td>THREADF</td>
<td>Number of threads used for function evaluation</td>
<td>1</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads used by Conopt internally</td>
<td>GAMS Threads</td>
</tr>
</tbody>
</table>

**cooptfile** *(string):* ↔

**Flg_2DDir** *(boolean):* Flag for computing and using directional 2nd derivatives. ↔

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off of the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see **Flg_Hessian**) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: auto

**Flg_Convex** *(boolean):* Flag for defining a model to be convex ↔

When turned on (the default is off) CONOPT knows that a local solution is also a global solution, whether it is optimal or infeasible, and it will be labeled appropriately. At the moment, Flg_NegCurve will be turned off. Other parts of the code will gradually learn to take advantage of this flag.

Default: 0

**Flg_Crash_Slack** *(boolean):* Flag for pre-selecting slacks for the initial basis. ↔

When turned on (1) CONOPT will select all infeasible slacks as the first part of the initial basis.

Default: 0

**Flg_DBG_Inv** *(boolean):* Flag for debugging interval evaluations. ↔
Flg_Dbg_Intv controls whether interval evaluations are debugged. Currently we check that
the lower bound does not exceed the upper bound for all intervals returned, both for function
values and for derivatives.

Default: 0

**Flg_Hessian (boolean):** Flag for computing and using 2nd derivatives as Hessian of Lagrangian.

If turned on, compute the structure of the Hessian of the Lagrangian and make it available
to CONOPT. The default is usually on, but it will be turned off if the model has external
equations (defined with =X=) or cone constraints (defined with =C=) or if the Hessian
becomes too dense. See also Flg_2DDir and HessianMemFac.

Default: auto

**Flg_Interv (boolean):** Flag for using intervals in the Preprocessor

If turned on (default), CONOPT will attempt to use interval evaluations in the preprocessor
to determine if functions are monotone or if intervals for some of the variables can be excluded
as infeasible.

Default: 1

**Flg_NegCurve (boolean):** Flag for testing for negative curvature when apparently optimal

When turned on (the default) CONOPT will try to identify directions with negative curvature
when the model appears to be optimal. The objective is to move away from saddlepoints. Can
be turned off when the model is known to be convex and cannot have negative curvature.

Default: 1

**Flg_NoPen (boolean):** Flag for allowing the Model without penalty constraints

When turned on (the default) CONOPT will create and solve a smaller model without the
penalty constraints and variables and the minimax constraints and variables if the remaining
constraints are infeasible in the initial point. This is often a faster way to start the solution
process.

Default: 1

**Flg_Prep (boolean):** Flag for using the Preprocessor

If turned on (default), CONOPT will use its preprocessor to try to determine pre- and
post-triangluar components of the model and find definitional constraints.

Default: 1

**Flg_Range (boolean):** Flag for identifying sets of ranged constraints

If turned on (default), CONOPT will as part of its preprocessor look for sets of parallel linear
constraints and turn each set into a single ranged constraints. There is currently a potential
problem with the duals on these constraints and if duals are important ranges can be turned
off with this flag.

Default: 1

**Flg_SLPMode (boolean):** Flag for enabling SLP mode.
If Flg.SLPMode is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.

Default: 1

**Flg.SQPMode (boolean):** Flag for enabling of SQP mode.

If Flg.SQPMode is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: 1

**Flg_Square (boolean):** Flag for Square System. Alternative to defining modeltype=CNS in GAMS

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: 0

**Flg.TraceCNS (boolean):** Flag for tracing a CNS solution.

When turned on the model must, for fixed value of the objective variable, be a CNS model and must satisfy the conditions of a CNS. The model is first solved as a CNS with the initial value of the objective fixed and the objective is then minimized or maximized subject to the CNS constraints.

Default: 0

**Frq_Log_Simple (integer):** Frequency for log-lines for non-SLP/SQP iterations.

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Default: auto

**Frq_Log_SlpSqp (integer):** Frequency for log-lines for SLP or SQP iterations.

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 50 constraints or 1000 variables.

Default: auto

**Frq_Rescale (integer):** Rescaling frequency.

The row and column scales are recalculated at least every Frq_Rescale new point (degenerate iterations do not count), or more frequently if conditions require it.

Default: 5

**HEAPLIMIT (real):** Maximum Heap size in MB allowed
Range: \([0, \infty)\]

Default: \(1e20\)

**HessianMemFac** (real): Memory factor for Hessian generation: Skip if Hessian elements > Nonlinear Jacobian elements*HessianMemFac, 0 means unlimited.

The Hessian of the Lagrangian is considered too dense therefore too expensive to evaluate and use, and it is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by HessianMemFac. See also Flg_Hessian. If HessianMemFac = 0.0 (the default value) then there is no limit on the number of Hessian elements.

The following cells are used to count calls of various routines and the time spend in them. The timing is usually only turned on if some debugging level is on and the statistics is only printed in these cases.

Default: 0

**Lim_Dbg_1Drv** (integer): Flag for debugging of first derivatives

Lim_Dbg_1Drv controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with \(=X=\). The amount of debugging is controlled by **Mtd_Dbg_1Drv**. See **Lim_Hess_Est** for a definition of when derivatives are considered wrong.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The derivatives are tested in the initial point only.</td>
</tr>
<tr>
<td>0</td>
<td>No debugging</td>
</tr>
<tr>
<td>+n</td>
<td>The derivatives are tested in all iterations that can be divided by Lim_Dbg_1Drv, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)</td>
</tr>
</tbody>
</table>

**Lim_Err_2DDir** (integer): Limit on errors in Directional Second Derivative evaluation.

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lim_Err_2DDir times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: 10

**Lim_Err_Fnc_Drv** (integer): Limit on number of function evaluation errors. Overwrites GAMS Domlim option

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lim_Err_Fnc_Drv times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: GAMS DomLim

**Lim_Err_Hessian** (integer): Limit on errors in Hessian evaluation.
If the evaluation of Hessian information has failed more than Lim_Err_Hessian times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: 10

**Lim_Hess_Est (real):** Upper bound on second order terms. →

The function and derivative debugger (see Lim_Dbgs_1Drv) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses Lim_Hess_Est as an estimate of the upper bound on second order derivatives in the model. Larger Lim_Hess_Est values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Default: 1.e4

**Lim_Iteration (integer):** Maximum number of iterations. Overwrites GAMS Iterlim option. ←

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option Lim_Time) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: GAMS IterLim

**Lim_Msg_Dbg_1Drv (integer):** Limit on number of error messages from function and derivative debugger. ←

The function and derivative debugger (see Lim_Dbgs_1Drv) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after Lim_Msg_Dbg_1Drv error(s) have been found.

Default: 10

**Lim_Msg_Large (integer):** Limit on number of error messages related to large function value and Jacobian elements. ←

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function values and derivatives. This bound is 1.e30. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating Jacobian elements, up to a limit of Lim_Msg_Large error messages.

Default: 10

**Lim_NewSuper (integer):** Maximum number of new superbasic variables added in one iteration. ←

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lim_NewSuper is positive then the limit is min(500,Lim_NewSuper). If Lim_NewSuper is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: auto
**Lim_Pre_Msg** (*integer*): Limit on number of error messages related to infeasible pre-triangle.

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than Lim_Pre_Msg elements the report is considered difficult to use and it is skipped.

Default: 25

**Lim_RedHess** (*integer*): Maximum number of superbasic variables in the approximation to the Reduced Hessian.

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on on the size. The limit is Lim_RedHess if Lim_RedHess is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Default: auto

**Lim_SlowPrg** (*integer*): Limit on number of iterations with slow progress (relative less than Tol_Obj_Change).

The optimization is stopped if the relative change in objective is less than Tol_Obj_Change for Lim_SlowPrg consecutive well-behaved iterations.

Default: 20

**Lim_StallIter** (*integer*): Limit on the number of stalled iterations.

An iteration is considered a stalled iteration if there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, especially if the model does not satisfy standard constraints qualifications and does not have a KKT point. To prevent cycling CONOPT will therefore stop after Lim_StallIter stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Default: 100

**Lim_Start_Degen** (*integer*): Limit on number of degenerate iterations before starting degeneracy breaking strategy.

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds Lim_Start_Degen CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: 100

**Lim_Time** (*real*): Time Limit. Overwrites the GAMS Reslim option.


The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Lim_Time is overwritten by Reslim when called from GAMS.

Range: \([0, \infty]\)

Default: GAMS ResLim

**Lim_Variable (real):** Upper bound on solution values and equation activity levels

If the value of a variable, including the objective function value and the value of slack variables, exceeds Lim_Variable then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded. A bound cannot exceed this value.

Range: \([1.e5, 1.e30]\)

Default: 1.e15

**Lim_Method (integer):** Method used to determine if and/or which Linear Feasibility Models to use

The Linear Feasibility Model can use different objectives: Objective 1 is no objective, i.e. the first point that satisfies the Linear Feasibility Model is used as a starting point for the Full Model. Objective 2 minimizes a scaled distance from the initial point for all variables defined by the modeler. Objective 3 minimizes a scaled distance from the initial point for all variables including those not defined by the modeler. Objective 4 minimizes a scaled distance from random a point selected away from bounds.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ignore Linear Feasibility Model in the first round and use objective 2, 3, and 4 (see above) in the following rounds as long as model is locally infeasible. This is the default method.</td>
</tr>
<tr>
<td>2</td>
<td>Use Linear Feasibility Model with objective 1 in the first round and continue with objective 2, 3, and 4 in the following rounds as long as model is locally infeasible.</td>
</tr>
<tr>
<td>3</td>
<td>Use Linear Feasibility Model with objective 2 in the first round and continue with objective 3 and 4 in the following rounds as long as model is locally infeasible.</td>
</tr>
</tbody>
</table>

**Mtd_Dbg_1Drv (integer):** Method used by the function and derivative debugger.

The function and derivative debugger (turned on with Lim_Dbg_1Drv) can perform a fairly cheap test or a more extensive test, controlled by Mtd_Dbg_1Drv. See Lim_Hess_Est for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.</td>
</tr>
<tr>
<td>1</td>
<td>As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used of the cheap test does not find an error but one is expected to exist.</td>
</tr>
</tbody>
</table>


**Mtd\_RedHess (integer):** Method for initializing the diagonal of the approximate Reduced Hessian

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by Mtd\_RedHess:

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.</td>
</tr>
<tr>
<td>1</td>
<td>The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.</td>
</tr>
</tbody>
</table>

**Mtd\_Scale (integer):** Method used for scaling.

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution (see also Frq\_Rescale.) The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see Tol\_Scale\_Max and Tol\_Scale\_Min. Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than Tol_Jac_Min are considered equal to Tol_Jac_Min.</td>
</tr>
<tr>
<td>1</td>
<td>Similar to 3 below, but the projection on the interval [Tol_Scale_Min,Tol_Scale_Max] is applied at a different stage. With method 1, abs(X)*abs(Jac) with small X and very large Jac is scaled very aggressively with a factor abs(Jac). With method 3, the scale factor is abs(X)*abs(Jac). The difference is seen in models with terms like Sqrt(X) close to X = 0.</td>
</tr>
<tr>
<td>2</td>
<td>As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.</td>
</tr>
<tr>
<td>3</td>
<td>Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by the maximum of the largest term and the value of the variable. A term is here defined as abs(X)*abs(Jac) where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factor are then projected on the interval between Tol_Scale_Min and Tol_Scale_Max.</td>
</tr>
</tbody>
</table>

**Mtd\_Step\_Phase0 (integer):** Method used to determine the step in Phase 0.

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by Mtd\_Step\_Phase0:
Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this method. It is the default method for optimization models.</td>
</tr>
<tr>
<td>1</td>
<td>A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending.</td>
</tr>
</tbody>
</table>

\textbf{Mtd\_Step\_Tight (integer):} Method used to determine the maximum step while tightening tolerances.

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by \texttt{Mtd\_Step\_Tight}:

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this method.</td>
</tr>
<tr>
<td>1</td>
<td>A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.</td>
</tr>
</tbody>
</table>

\textbf{Num\_Rounds (integer):} Number of rounds with Linear Feasibility Model

Lin\_Method defined which Linear Feasibility Model are going to be solved if the previous models end Locally Infeasible. The number of rounds is limited by \texttt{Num\_Rounds}.

Range: \{1, \ldots, 4\}

Default: 4

\textbf{Rat\_NoPen (real):} Limit on ratio of penalty constraints for the No\_Penalty model to be solved

The No-Penalty model can only be generated and solved if the number of penalty and minimax constraints exceed \texttt{Rat\_NoPen} times the constraints in the Full Model.

Default: 0.1

\textbf{THREAD2D (integer):} Number of threads used for second derivatives

Range: \{0, \ldots, \infty\}

Default: 1

\textbf{THREADC (integer):} Number of compatibility threads used for comparing different values of THREADS

Range: \{0, \ldots, \infty\}

Default: 1
**ThreadF (integer):** Number of threads used for function evaluation

Range: \{0, ..., \infty\}

Default: 1

**Threads (integer):** Number of threads used by Conopt internally

Range: \{0, ..., \infty\}

Default: GAMS Threads

**Tol_Bound (real):** Bound filter tolerance for solution values close to a bound.

A variable is considered to be at a bound if its distance from the bound is less than \(\text{Tol\_Bound} \times \text{Max}(1, \text{ABS}(\text{Bound}))\). The tolerance is used to build the initial bases and is used to flag variables during output.

Range: \([3.e-13, 1.e-5]\]

Default: 1.e-7

**Tol_BoxSize (real):** Initial box size for trust region models for overall model

The new Phase 0 methods solves an LP model based on a scaled and linearized version of the model with an added trust region box constraint around the initial point. \(\text{Tol\_BoxSize}\) defines the size of the initial trust region box. During the optimization the trust region box is adjusted based on how well the linear approximation fits the real model.

Range: \([0.01, 1.e6]\]

Default: 10

**Tol_BoxSize_Lin (real):** Initial box size for trust region models for linear feasibility model

Similar to \(\text{Tol\_BoxSize}\) but applied to the linear feasibility model. Since this model has linear constraints the default initial box size is larger.

Range: \([0.01, 1.e8]\]

Default: 1000

**Tol_BoxLinFac (real):** Box size factor for linear variables applied to trust region box size

The trust region box used in the new Phase 0 method limits the change of variables so 2nd order terms will not become too large. Variables that appear linearly do not have 2nd order terms and the initial box size is therefore larger by a factor \(\text{Tol\_BoxLinFac}\).

Parameters related to growth factors and initial values in the definitional constraints

Range: \([1, 1.e4]\]

Default: 10

**Tol_Def_Mult (real):** Largest growth factor allowed in the block of definitional constraints
The block of definitional constraints form a triangular matrix. This triangular matrix can hide large accumulating growth factors that can lead to increases in the initial sum of infeasibilities and to numerical instability. Tol_Def_Mult is an upper bound on these growth factors. If it is exceeded some critical chains of definitional constraints will be broken leading to a larger internal model, that should be numerically better behaved.

Parameters related to scaling

Range: \([1.01, \infty]\)

Default: 1.e4

**Tol_Feas_Max** *(real)*: Maximum feasibility tolerance (after scaling). \(\leftarrow\)

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. Tol_Feas_Max is the upper bound on the dynamic feasibility tolerance and **Tol_Feas_Min** is the lower bound. It is NOT recommended to use loose feasibility tolerances since the objective, including the sum of infeasibility objective, will be less accurate and it may prevent convergence.

Range: \([1.e-10, 1.e-3]\)

Default: 1.e-7

**Tol_Feas_Min** *(real)*: Minimum feasibility tolerance (after scaling). \(\leftarrow\)

See **Tol_Feas_Max** for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: \([3.e-13, 1.e-5]\)

Default: 4.e-10

**Tol_Feas_Tria** *(real)*: Feasibility tolerance for triangular equations. \(\leftarrow\)

Triangular equations are usually solved to an accuracy of **Tol_Feas_Min**. However, if a variable reaches a bound or if a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to Tol_Feas_Tria.

Range: \([3.e-13, 1.e-4]\)

Default: 1.0e-8

**Tol_Fixed** *(real)*: Tolerance for defining variables as fixed based on initial or derived bounds. \(\leftarrow\)

A variable is considered fixed if the distance between the bounds is less than Tol_Fixed \(*\ Max(1, Abs(Bound))\). The tolerance is used both on the users original bounds and on the derived bounds that the preprocessor implies from the constraints of the model.

Accuracies for linesearch and updates

Range: \([3.e-13, 1.e-8]\)

Default: 4.e-10

**Tol_Jac_Min** *(real)*: Filter for small Jacobian elements to be ignored during scaling. \(\leftarrow\)
A Jacobian element is considered insignificant if it is less than Tol_Jac_Min. The value is used to select which small values are scaled up during scaling of the Jacobian. It is only used with scaling method Mtd_Scale = 0.

Range: $[1.e^{-7}, 1.e^{-3}]$

Default: $1.e^{-5}$

**Tol_Linesearch** *(real)*: Accuracy of One-dimensional search.

The onedimensional search is stopped if the expected decrease in the objective estimated from a quadratic approximation is less than Tol_Linesearch times the decrease so far in this onedimensional search.

Range: $[0.05, 0.8]$  

Default: 0.2

**Tol_Obj_Acc** *(real)*: Relative accuracy of the objective function.

It is assumed that the objective function can be computed to an accuracy of Tol_Obj_Acc * max( 1, abs(Objective) ). Smaller changes in objective are considered to be caused by round-off errors.

Range: $[3.0e^{-14}, 10.e^{-6}]$

Default: $3.0e^{-13}$

**Tol_Obj_Change** *(real)*: Limit for relative change in objective for well-behaved iterations.

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than Tol_Obj_Change * Max(1,Abs(Objective)). See also Lim_SlowPrg.

Range: $[3.0e^{-13}, 1.0e^{-5}]$

Default: $3.0e^{-12}$

**Tol_Optimality** *(real)*: Optimality tolerance for reduced gradient when feasible.

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than Tol_Optimality.

Range: $[3.e^{-13}, 1]$

Default: $1.e^{-7}$

**Tol_Opt_Infeas** *(real)*: Optimality tolerance for reduced gradient when infeasible.

The reduced gradient is considered zero and the solution infeasible if the largest superbasic component of the reduced gradient is less than Tol_Opt_Infeas.

Range: $[3.e^{-13}, 1]$

Default: $1.e^{-7}$

**Tol_Opt_LinF** *(real)*: Optimality tolerance when infeasible in Linear Feasibility Model
This is a special optimality tolerance used when the Linear Feasibility Model is infeasible. Since the model is linear the default value is smaller than for nonlinear submodels.

Pivot tolerances

Range: \([3.e-13, 1.e-4]\)

Default: \(1.e-10\)

**Tol_Piv_Abs** *(real)*: Absolute pivot tolerance. \(\leftarrow\)

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Tol_Piv_Abs. There is also a relative test, see **Tol_Piv_Rel**.

Range: \([2.2e-16, 1.e-7]\)

Default: \(1.e-10\)

**Tol_Piv_Abs_Ini** *(real)*: Absolute Pivot Tolerance for building initial basis. \(\leftarrow\)

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: \([3.e-13, 1.e-3]\)

Default: \(1.e-7\)

**Tol_Piv_Abs_NLTr** *(real)*: Absolute pivot tolerance for nonlinear elements in pre-triangular equations. \(\leftarrow\)

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is Tol_Piv_Abs. The value cannot be less than Tol_Piv_Abs.

Range: \([2.2e-16, 1.e-3]\)

Default: \(1.e-5\)

**Tol_Piv_Ratio** *(real)*: Relative pivot tolerance during ratio-test \(\leftarrow\)

During ratio-tests, the lower bound on the slope of a basic variable to potentially leave the basis is Tol_Piv_Ratio * the largest term in the computation of the tangent.

Range: \([1.e-10, 1.e-6]\)

Default: \(1.e-8\)

**Tol_Piv_Rel** *(real)*: Relative pivot tolerance during basis factorizations. \(\leftarrow\)

During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least Tol_Piv_Rel * the largest absolute value in the column. Small values or Tol_Piv_Rel will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of Tol_Piv_Rel may help.

Range: \([1.e-3, 0.9]\)

Default: \(0.05\)
**Tol_Piv_Rel_Init** *(real)*: Relative Pivot Tolerance for building initial basis

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: \([1.e^{-4}, 0.9]\]

Default: \(1.e^{-3}\)

**Tol_Piv_Rel_Updt** *(real)*: Relative pivot tolerance during basis updates.

During basis changes CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least \(\text{Tol}_{\text{Piv Rel Updt}} \times \text{the other element}\). Smaller values of \(\text{Tol}_{\text{Piv Rel Updt}}\) will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: \([1.e^{-3}, 0.9]\]

Default: 0.05

**Tol_Scale2D_Min** *(real)*: Lower bound for scale factors based on large 2nd derivatives.

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than \(\text{Tol}_{\text{Scale Min}}\). CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than \(\text{Tol}_{\text{Scale2D Min}}\).

Range: \([1.e^{-9}, 1]\]

Default: 1.e-6

**Tol_Scale_Max** *(real)*: Upper bound on scale factors.

Scale factors are projected on the interval from \(\text{Tol}_{\text{Scale Min}}\) to \(\text{Tol}_{\text{Scale Max}}\). Is used to prevent very large or very small scale factors due to pathological types of constraints. The upper limit is selected such that \(\text{Square(X)}\) can be handled for \(X\) close to \(\text{Lim Variable}\). More nonlinear functions may not be scalable for very large variables.

Range: \([1, 1.e30]\]

Default: 1.e25

**Tol_Scale_Min** *(real)*: Lower bound for scale factors computed from values and 1st derivatives.

Scale factors used to scale variables and equations are projected on the range \(\text{Tol}_{\text{Scale Min}}\) to \(\text{Tol}_{\text{Scale Max}}\). The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for \(\text{Tol}_{\text{Scale Min}}\) is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of \(\text{Tol}_{\text{Scale Min}} < 1\).

Range: \([1.e^{-10}, 1]\]

Default: 1

**Tol_Scale_Var** *(real)*: Lower bound on \(x\) in \(x\times Jac\) used when scaling.
Rows are scaled so the largest term $x \cdot \text{Jac}$ is around 1. To avoid difficulties with models where \text{Jac} is very large and $x$ very small a lower bound of $\text{Tol\_Scale\_Var}$ is applied to the $x$-term.

Largest Jacobian element and tolerance in 2nd derivative tests:

Range: $[1.e^{-8}, 1]$  
Default: $1.e^{-5}$

**Tol\_Zero** *(real)*: Zero filter for Jacobian elements and inversion results.  
Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than $\text{Tol\_Piv\_Abs}/10$.

Default: $1.e^{-20}$

**Trace\_MinStep** *(real)*: Minimum step between Reinversions when using TraceCNS.  
The optimization is stopped with a slow convergence message if the change in trace variable or objective is less than this tolerance between reinversions for more than two consecutive reinversions. The step is scaled by the distance from the initial value to the critical bound.

Range: $[0, 1]$  
Default: 0.001

### 5.9 CONVERT

#### 5.9.1 Introduction

CONVERT is a utility which transforms a GAMS model instance into a scalar model where all confidential information has been removed or into formats used by other modeling and solution systems. CONVERT is designed to achieve the following goals:

- Permit users to convert a confidential model into GAMS scalar format so that any identifiable structure is removed. It can then be passed on to others for investigation without confidentiality being lost.
- A way of sharing GAMS test problems for use with other modeling systems or solvers.

CONVERT comes free of charge with any licensed GAMS system and can convert GAMS models into a number of formats, see Section Target languages and formats for a list.

#### 5.9.2 How to use CONVERT

CONVERT is run like any other GAMS solver. From the command line this is:

```
>> gams modelname modeltype=convert
```

where *modelname* is the GAMS model name and *modeltype* the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). CONVERT can also be specified via the option statement within the model itself before the solve statement:

```
option modeltype=convert;
```
5.9.3 The GAMS Scalar Format

By default, CONVERT generates a scalar GAMS model \((\text{gams.gms})\) from the input model. The scalar model exhibits the following characteristics:

- A model without sets or indexed parameters. It does not exhibit any of the advanced characteristics of modeling systems and is easily transformable.
- A model with a new set of individual variables, depicting each variable in the GAMS model as one of 3 types: positive, integer or binary. Each variable is numbered sequentially, i.e. all positive GAMS variables are mapped into single variables \(x_1, x_2, \ldots, x_n\).
- A model with individual equations depicting each variable in the GAMS model. All equations are also numbered sequentially, that is equations \(e_1, e_2, \ldots, e_m\).

Equation and variable bounds, as well as variable starting values are preserved from the original GAMS formulation.

As an example, suppose the user wishes to translate the GAMS Model Library model \texttt{trnsport}\) into scalar format. One would run

\begin{verbatim}
gams trnsport.gms lp=convert
\end{verbatim}

which would generate the following scalar model \texttt{gams.gams}:

\begin{verbatim}
* LP written by GAMS Convert at 11/19/20 15:28:05
*
* Equation counts
* Total    E    G    L    N    X    C    B
*       6    1    3    2    0    0    0    0
*
* Variable counts
*                        x   b  i  s1s  s2s   sc   si  sint
*      Total   cont binary integer sos1 sos2 scont sint
*            7    7    0    0    0    0    0    0
*   FX        0
*
* Nonzero counts
*   Total const   NL
*        19    19    0
*
* Solve m using LP minimizing x7;

Variables
\quad x1, x2, x3, x4, x5, x6, x7;

Positive Variables
\quad x1, x2, x3, x4, x5, x6;

Equations
\quad e1, e2, e3, e4, e5, e6;

\begin{verbatim}
e1..  -0.225 * x1 - 0.153 * x2 - 0.162 * x3 - 0.225 * x4 - 0.162 * x5 - 0.126 * x6 + x7 =E= 0;
e2..  x1 + x2 + x3 =L= 350;
\end{verbatim}
\end{verbatim}
5.9 CONVERT

\[ e_3.. \ x_4 + x_5 + x_6 =L= 600; \]
\[ e_4.. \ x_1 + x_4 =G= 325; \]
\[ e_5.. \ x_2 + x_5 =G= 300; \]
\[ e_6.. \ x_3 + x_6 =G= 275; \]

Model m / all /;

m.limrow = 0;

m.limcol = 0;

Solve m using LP minimizing x_7;

Note that the resulting scalar model does not contain any of the descriptive information about the data or the context of the constraints.

Additionally, a dictionary file (dict.txt) is created by default which specifies a mapping between the variable and equation names in the scalar model and their corresponding names in the original model.

For the above example, the dictionary file is

LP written by GAMS Convert at 11/19/20 15:28:05

Equation counts

<table>
<thead>
<tr>
<th>Total</th>
<th>E</th>
<th>G</th>
<th>L</th>
<th>N</th>
<th>X</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Variable counts

<table>
<thead>
<tr>
<th>x</th>
<th>b</th>
<th>i</th>
<th>sis</th>
<th>s2s</th>
<th>sc</th>
<th>si</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Nonzero counts

<table>
<thead>
<tr>
<th>Total</th>
<th>const</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

Equations 1 to 6
\[ e_1 \ \text{cost} \]
\[ e_2 \ \text{supply(seattle)} \]
\[ e_3 \ \text{supply(san-diego)} \]
\[ e_4 \ \text{demand(new-york)} \]
\[ e_5 \ \text{demand(chicago)} \]
\[ e_6 \ \text{demand(topeka)} \]

Variables 1 to 7
\[ x_1 \ \text{x(seattle,new-york)} \]
\[ x_2 \ \text{x(seattle,chicago)} \]
\[ x_3 \ \text{x(seattle,topeka)} \]
\[ x_4 \ \text{x(san-diego,new-york)} \]
\[ x_5 \ \text{x(san-diego,chicago)} \]
\[ x_6 \ \text{x(san-diego,topeka)} \]
\[ x_7 \ z \]

Conversion of a GAMS model to a scalar one may be handy for model debugging. However, in this case, it may be good to retain the original variable and equation names. The following simple sed command attempts to achieve this:
For the above example, this outputs:

**Variables**

\( x(\text{seattle,new-york}), x(\text{seattle,Chicago}), x(\text{seattle,topeka}), x(\text{san-diego,new-york}), x(\text{san-diego,Chicago}) \)

**Positive Variables**

\( x(\text{seattle,new-york}), x(\text{seattle,Chicago}), x(\text{seattle,topeka}), x(\text{san-diego,new-york}), x(\text{san-diego,Chicago}) \)

**Equations**

\[
\begin{align*}
\text{cost} & : -0.225 \times x(\text{seattle,new-york}) - 0.153 \times x(\text{seattle,Chicago}) - 0.162 \times x(\text{seattle,topeka}) - 0.225 \times x(\text{san-diego,new-york}) - 0.162 \times x(\text{san-diego,Chicago}) - 0.126 \times x(\text{san-diego,topeka}) + z = 0; \\
\text{supply(\text{seattle})} & : x(\text{seattle,new-york}) + x(\text{seattle,Chicago}) + x(\text{seattle,topeka}) = 350; \\
\text{supply(\text{san-diego})} & : x(\text{san-diego,new-york}) + x(\text{san-diego,Chicago}) + x(\text{san-diego,topeka}) = 600; \\
\text{demand(\text{new-york})} & : x(\text{seattle,new-york}) + x(\text{san-diego,new-york}) = 325; \\
\text{demand(\text{Chicago})} & : x(\text{seattle,Chicago}) + x(\text{san-diego,Chicago}) = 300; \\
\text{demand(\text{topeka})} & : x(\text{seattle,topeka}) + x(\text{san-diego,topeka}) = 275;
\end{align*}
\]

Of course, this is not a valid GAMS code and cannot be compiled, but it may be sufficient to view the model algebra as generated by the GAMS compiler.

By using

\[
\begin{align*}
\text{sed } -n -e "s:~ *((exbi)[0-9][0-9]*\}) \((.*):s/1/2/g:gp" \text{dict.txt } | \text{sed } -n '1!G;h;$p' > \text{mod.txt} \\
\text{sed -f mod.txt gams.gms}
\end{align*}
\]

one gets for this example

**Variables**

\( x_{\text{seattle,new-york}}, x_{\text{seattle,Chicago}}, x_{\text{seattle,topeka}}, x_{\text{san-diego,new-york}}, x_{\text{san-diego,Chicago}} \)

**Positive Variables**

\( x_{\text{seattle,new-york}}, x_{\text{seattle,Chicago}}, x_{\text{seattle,topeka}}, x_{\text{san-diego,new-york}}, x_{\text{san-diego,Chicago}} \)

**Equations**

\[
\begin{align*}
\text{cost} & : -0.225 \times x_{\text{seattle,new-york}} - 0.153 \times x_{\text{seattle,Chicago}} - 0.162 \times x_{\text{seattle,topeka}} - 0.225 \times x_{\text{san-diego,new-york}} - 0.162 \times x_{\text{san-diego,Chicago}} - 0.126 \times x_{\text{san-diego,topeka}} + z = 0; \\
\text{supply(\text{seattle})} & : x_{\text{seattle,new-york}} + x_{\text{seattle,Chicago}} + x_{\text{seattle,topeka}} = 350; \\
\text{supply(\text{san-diego})} & : x_{\text{san-diego,new-york}} + x_{\text{san-diego,Chicago}} + x_{\text{san-diego,topeka}} = 600; \\
\text{demand(\text{new-york})} & : x_{\text{seattle,new-york}} + x_{\text{san-diego,new-york}} = 325; \\
\text{demand(\text{Chicago})} & : x_{\text{seattle,Chicago}} + x_{\text{san-diego,Chicago}} = 300; \\
\text{demand(\text{topeka})} & : x_{\text{seattle,topeka}} + x_{\text{san-diego,topeka}} = 275;
\end{align*}
\]

This can even be compiled by GAMS and gives the correct solution.

The proposed commands come with several limitations and may not produce in all cases the desired output. For example, wrong results would be printed if the original model contains variable or equation names that start with \{b,i,e,x\} [digit]. Also semicontinuous or semiinteger variables or special ordered sets are not supported by the above. We leave it to the experienced user to extend the command appropriately.
5.9.4 The OSiL Format

The Optimization Services Instance Language (OSiL) [70] specifies an XML-based format to represent optimization problem instances. GAMS/CONVERT can write MINLP model instances in OSiL format. Expression trees are written in OSnL format.

Next to the indexed operations for sum, product, minimum, and maximum, and the operations for subtraction and division, the following intrinsic functions are mapped to their OSnL counterparts: \texttt{sqr}, \texttt{sqrt}, \texttt{exp}, \texttt{log}, \texttt{log2}, \texttt{abs}, \texttt{cos}, \texttt{sin}, \texttt{tan}, \texttt{arccos}, \texttt{arctan}, \texttt{sinh}, \texttt{cosh}, \texttt{tanh}, \texttt{pi}, \texttt{div}, \texttt{gamma}, \texttt{loggamma}, \texttt{floor}, \texttt{ceil}, \texttt{round}, \texttt{trunc}, \texttt{sign fact}, \texttt{binomial}. The functions \texttt{cvPower}, \texttt{power}, \texttt{rpower}, \texttt{vcpower} are all mapped to OSnL's power operator, thus conditions on arguments are not preserved. Functions \texttt{arctan2}, \texttt{centropy}, \texttt{edist}, \texttt{erf}, and \texttt{poly} are represented by an expression according to their algebraic definition. The intrinsic functions \texttt{signpower}, \texttt{entropy}, \texttt{sigmoid}, \texttt{gammareg}, \texttt{beta}, \texttt{logbeta}, and \texttt{betareg} are also written to OSiL files, but do not follow the OSnL standard (as it currently does not offer these functions). Thus, OSiL readers may reject XML files that use these functions. Finally, also the logical functions are written to OSiL by using their OSnL counterpart.

5.9.5 User-Specified Options

CONVERT options are passed on through option files. If you specify \texttt{<modelname>.optfile = 1;} before the SOLVE statement in your GAMS model, CONVERT will look for and read an option file with the name \texttt{convert.opt} (see The Solver Options File for general use of solver option files). The syntax for the CONVERT option file is

\begin{verbatim}
 optedname value
\end{verbatim}

with one option on each line. For example,

\begin{verbatim}
ampl
\end{verbatim}

This option file would tell CONVERT to produce an AMPL input file. For file format options, the user can specify the filename for the file to be generated. For example, the option file entry

\begin{verbatim}
lingo myfile.lng
\end{verbatim}

would generate a LINGO input file format called \texttt{myfile.lng}. Using the option \texttt{lingo} by itself, would produce the default output file for that option (\texttt{lingo.lng}).

All available options are listed in the following tables.
### 5.9.5.1 Target languages and formats

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Generates all supported file formats</td>
<td></td>
</tr>
<tr>
<td>Ampl</td>
<td>Generates Ampl input file</td>
<td>ampl.mod</td>
</tr>
<tr>
<td>CplexLP</td>
<td>Generates CPLEX LP format input file</td>
<td>cplex.lp</td>
</tr>
<tr>
<td>CplexMPS</td>
<td>Generates CPLEX MPS format input file</td>
<td>cplex.mps</td>
</tr>
<tr>
<td>Dict</td>
<td>Generates Convert to GAMS Dictionary</td>
<td>dict.txt</td>
</tr>
<tr>
<td>DictMap</td>
<td>Generates Convert to GAMS Dictionary Map</td>
<td>dictmap.gdx</td>
</tr>
<tr>
<td>DumpGDX</td>
<td>Generates GDX with model data incl. Jacobian and Hessian evaluated at current point</td>
<td>dump.gdx</td>
</tr>
<tr>
<td>FileList</td>
<td>Generates file list of file formats generated</td>
<td>files.gdx</td>
</tr>
<tr>
<td>FixedMPS</td>
<td>Generates fixed format MPS file</td>
<td>fixed.mps</td>
</tr>
<tr>
<td>Gams</td>
<td>Generates GAMS scalar model. This is the default conversion format used.</td>
<td>gams.gms</td>
</tr>
<tr>
<td>JuMP</td>
<td>Generates JuMP scalar model</td>
<td>jump.jl</td>
</tr>
<tr>
<td>Lingo</td>
<td>Generates Lingo input file</td>
<td>lingo.lng</td>
</tr>
<tr>
<td>NLP2dual</td>
<td>Generates the Wolfe dual of a smooth optimization model</td>
<td>gamsdual.lng</td>
</tr>
<tr>
<td>NLP2MCP</td>
<td>Generates GAMS scalar MCP model</td>
<td>gamsmcp.gms</td>
</tr>
<tr>
<td>OSiL</td>
<td>Generates Optimization Services instance Language (OSiL) file</td>
<td>osil.xml</td>
</tr>
<tr>
<td>Pyomo</td>
<td>Generates Pyomo Concrete scalar model</td>
<td>pyomo.py</td>
</tr>
</tbody>
</table>

### 5.9.5.2 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DualType</td>
<td>Controls type of Wolfe dual to generate in NLP2dual</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>None:</strong> No Wolfe dual generated</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NLPScalarBounds:</strong> NLP dual where variable bounds become scalars used in equations</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NLPConstantBounds:</strong> NLP dual where finite variable bounds become constants in equations</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>BiLevel:</strong> Bilevel model with outer problem optimizing over the duals</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>MPEC:</strong> MPEC obtained by explicitly including FOC of BiLevel inner problem</td>
<td></td>
</tr>
<tr>
<td>GDXHessian</td>
<td>Enable hessian information for DumpGDX</td>
<td>0</td>
</tr>
<tr>
<td>GDXNames</td>
<td>Enable variable and equation names for DumpGDX</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>GDXQuadratic</td>
<td>Enable quadratic information for DumpGDX</td>
<td>0</td>
</tr>
<tr>
<td>GDXUELs</td>
<td>Enable UELs for DumpGDX</td>
<td>1</td>
</tr>
<tr>
<td>GmsInsert</td>
<td>Line to be inserted before the solve statement</td>
<td>$\text{if NOT } '\text{gams.u1} == ''' \text{include } '\text{gams.u1}'$</td>
</tr>
<tr>
<td>HeaderTime Stamp</td>
<td>Control format of time stamp in header of output file</td>
<td>default</td>
</tr>
<tr>
<td>Interval Eval</td>
<td>Include interval evaluations in DumpGDX</td>
<td>0</td>
</tr>
<tr>
<td>ObjVar</td>
<td>Name of objective variable</td>
<td>GAMS index name, e.g. x1</td>
</tr>
<tr>
<td>Permute Equs</td>
<td>Random seed for permutation of equations (0: no permutation)</td>
<td>0</td>
</tr>
<tr>
<td>Permute Vars</td>
<td>Random seed for permutation of variables (0: no permutation)</td>
<td>0</td>
</tr>
<tr>
<td>QExtract Alg</td>
<td>Quadratic extraction algorithm in GAMS interface</td>
<td>0</td>
</tr>
<tr>
<td>Reform</td>
<td>Force reformulations</td>
<td>100</td>
</tr>
<tr>
<td>SkipNRows</td>
<td>Skip constraints of type =N=</td>
<td>0</td>
</tr>
<tr>
<td>Width</td>
<td>Max line width of output format</td>
<td>80</td>
</tr>
</tbody>
</table>

### 5.10 COPT

**Cardinal Optimizer** by Cardinal Operations is a solver for linear programming problems (LPs) and mixed-integer linear programming problems (MIPs).

#### 5.10.1 Usage

A GAMS/COPT or GAMS/COPT-Link license is required to use the GAMS/COPT interface that is distributed with GAMS. If only a GAMS/COPT-Link license is available, also a separate COPT license needs to be installed on the system, see the "Cardinal Operations User Guide" for details.

The following statement can be used inside your GAMS program to specify using COPT
Option MIP = COPT; { or LP, RMIP }

The above statement should appear before the Solve statement. If COPT was specified as the default solver during GAMS installation, the above statement is not necessary.

COPT supports special ordered sets (SOS), but no semicontinuous or semiinteger variables.

5.10.1.1 Specification of COPT Options

The following GAMS parameters are currently supported by GAMS/COPT: reslim, iterlim, nodlim, optcr, optca, tryint, and threads.

Further options can be set via a solver options file, see Section The Solver Option File for further information. Following is an example options file **copt.opt**:

```
TreeCutLevel 0
SimplexThreads 4
```

It will cause COPT to use cut generators only in the root node (for a MIP solve) and specifies to use 4 threads in the simplex algorithm.

5.10.1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to COPT via a GAMS/COPT solver options file, see Indicator Constraints for more details on its syntax.

5.10.1.3 MIP Starting Point

When solving a MIP, by default all values that are specified as variable levels in the GAMS model are passed as starting point to COPT (unless MipStartMode = 0). However, if GAMS option tryint is set to a nonzero value or MipStartMode = 2, then only values of binary and integer variables which fractionality is at most the value tryint are passed as starting point to COPT (possible fractional values are rounded to integers). COPT will then try to complete this partial solution to a feasible solution.

5.10.1.4 Solution Pool

When COPT solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option solnpooldump is specified, then all alternative solutions found by COPT are written into GDX files and an index file with the name given by the this option is written. If the option solnpooolmerge is specified, then all alternative solutions found by COPT are written into a single GDX file, which name is given by the this option.

The GAMS testlib model **dumpsol** shows an example use for this option.
5.10.1.5 Infeasible and unbounded LPs

If an LP is found infeasible by COPT, a primal infeasibility certificate in form of a Farkas proof (see also Mosek manual) is computed. The GAMS/COPT link returns the values of this certificate in the equations marginal values and sets the INFES markers (see solution listing) for those equations that are included in the Farkas proof.

If an LP is found unbounded by COPT, a dual infeasibility certificate in form of a primal ray is computed. If the problem is feasible, then the primal ray gives a direction in which the objective function can be infinitely improved. The GAMS/COPT link returns the values of the primal ray in the variable level values and sets UNBND markers (see solution listing) for those variables that are included in the ray.

COPT option ReqFarkasRay can be used to disable the calculation of these infeasibility certificates.

5.10.1.6 Finding an Irreducible Inconsistent Subsystem (IIS) of Constraints

If COPT declares an LP or MIP as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. Option iis can be used to enable finding an IIS.

As an example, consider a GAMS model containing

Positive Variables x, y;
Equation e1;
e1.. x + y =l= -1;

The corresponding COPT output when iis has been enabled will look like

Starting the simplex solver using 1 thread

Method Iteration Objective Primal.NInf Dual.NInf Time
Dual 0 0.0000000000e+00 1 0 0.00s
Solving finished
Status: Infeasible Objective: - Iterations: 0 Time: 0.00s
Start the IIS computation for an LP

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Min RowBnd</th>
<th>Max RowBnd</th>
<th>Min ColBnd</th>
<th>Max ColBnd</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.00s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.00s</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.00s</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.00s</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0.00s</td>
</tr>
</tbody>
</table>

IIS summary: 1 rows, 2 bounds of columns
IIS computation finished (0.000s)
IIS is minimal
Number of equations in IIS: 1
  Upper: e1 <= -1
Number of variables in IIS: 2
  Lower: x >= 0
  Lower: y >= 0

That is, COPT finds the problem infeasible. Then the IIS computation is started and an IIS is found. The IIS is then printed to the log and listing file. It consists of the lower bounds of variables x and y and equation e1. This suggests that the entire model can be made feasible by lowering the lower bound of x or y or by increasing the right-hand side of e1.
5.10.1.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasRelax parameter in a COPT solver option file.

With the FeasRelax option COPT selectively relaxes the variable bounds and constraint sides of an infeasible model instance in a way that a weighted penalty function is minimized. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

Attention

At the moment, COPT can only relax sides of linear constraints. Indicator and quadratic constraints are not considered for relaxation.

By default all equation sides are candidates for relaxation and weighted equally but none of the variable bounds can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the dot option feaspref. A zero preference means that the associated bound or constraint side is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given variable bound or constraint side will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a COPT solver option file. The syntax is:

(variable or equation).feaspref(value)

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the copt.opt file can be specified by:

```
fearsrelax 1
v.feaspref 1
v.feaspref('i1','*') 2
v.feaspref('i1','j2') 0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables v(i,j) have preference of 1, except variables over set element i1, which have a preference of 2. The variable over set element i1 and j2 has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords variables or equations instead of the individual variable and equations names (e.g. variables.feaspref 1).

The parameter FeasRelaxMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasRelaxMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations). Also check model feasopt1 in the GAMS model library.
5.10 COPT

5.10.2 List of COPT Options

In the following, we give a detailed list of available COPT options.

5.10.2.1 Limits and Tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarIterLimit</td>
<td>Barrier iteration limit</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>DualTol</td>
<td>The tolerance for dual solutions and reduced cost</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-09, 0.0001]</td>
<td></td>
</tr>
<tr>
<td>FeasTol</td>
<td>The feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-09, 0.1]</td>
<td></td>
</tr>
<tr>
<td>IntTol</td>
<td>The integer feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-09, 0.1]</td>
<td></td>
</tr>
<tr>
<td>MatrixTol</td>
<td>The input matrix coefficient tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1e-07]</td>
<td></td>
</tr>
<tr>
<td>NodeLimit</td>
<td>Limit of nodes for MIP</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>RelGap</td>
<td>The relative gap for MIP</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>TimeLimit</td>
<td>Time limit of the optimization</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1e+20]</td>
<td></td>
</tr>
</tbody>
</table>

5.10.2.2 Presolving and Scaling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dualize</td>
<td>Whether to dualize a problem before solving it</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Choose automatically. 0: No dualizing. 1: Dualizing the problem.</td>
<td></td>
</tr>
<tr>
<td>Presolve</td>
<td>Whether to perform presolving before solving a problem</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Choose automatically. 0: No presolving. 1: Presolving the problem.</td>
<td></td>
</tr>
<tr>
<td>Scaling</td>
<td>Whether to perform scaling before solving a problem</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Choose automatically. 0: No scaling. 1: Apply scaling.</td>
<td></td>
</tr>
</tbody>
</table>

5.10.2.3 LP solving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarHomogeneous</td>
<td>Whether to use homogeneous self-dual form in barrier</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Choose automatically. 0: No. 1: Yes.</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>BarOrder</td>
<td>Ordering method for barrier -1: Choose automatically.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: Approximate Minimum Degree (AMD).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Nested Dissection (ND).</td>
<td></td>
</tr>
<tr>
<td>Crossover</td>
<td>Whether to run crossover after barrier</td>
<td>1</td>
</tr>
<tr>
<td>DualPerturb</td>
<td>Whether to allow the objective function perturbation -1: Choose automatically</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: No perturbation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Allow objective function perturbation</td>
<td></td>
</tr>
<tr>
<td>DualPrice</td>
<td>Specifies the dual simplex pricing algorithm -1: Choose automatically.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: Using Devex pricing algorithm.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Using dual steepest-edge pricing algorithm</td>
<td></td>
</tr>
<tr>
<td>LpMethod</td>
<td>Specifies the LP method -1: Choose automatically.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>1: Dual simplex.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Barrier.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Crossover.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: Concurrent (Use simplex and barrier simultaneously).</td>
<td></td>
</tr>
<tr>
<td>ReqFarkasRay</td>
<td>Whether to return a Farkas or Ray when an LP is infeasible or unbounded</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.10.2.4 MIP solving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConflictAnalysis</td>
<td>Whether to perform conflict analysis -1: Automatic.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: No.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Yes.</td>
<td></td>
</tr>
<tr>
<td>CutLevel</td>
<td>Level of cutting planes generation -1: Automatic.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>DivingHeurLevel</td>
<td>Level of diving heuristics -1: Automatic.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>HeurLevel</td>
<td>Level of heuristics -1: Automatic.</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
</tbody>
</table>
### 5.10 COPT

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MipStartMode</td>
<td>Specifies MIP start mode</td>
<td>-1, if GAMS tryint = 0, otherwise 2</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Disable MIP starts.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Only use full and feasible MIP starts.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Try to complete partial feasible MIP starts by solving subMIPs.</td>
<td></td>
</tr>
<tr>
<td>MipStartNodeLimit</td>
<td>Limit of nodes for MIP start sub-MIP (to complete partial MIP starts)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1: unlimited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>NodeCutRounds</td>
<td>Maximum cut rounds in a local node</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>RootCutLevel</td>
<td>Level of root cutting planes generation</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>RootCutRounds</td>
<td>Maximum cut rounds in the root (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>RoundingHeurLevel</td>
<td>Level of rounding heuristics</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>StrongBranching</td>
<td>Level of strong branching</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>SubMipHeurLevel</td>
<td>Level of sub-MIP heuristics</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
<tr>
<td>TreeCutLevel</td>
<td>Level of tree cutting planes generation</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1: Automatic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Off.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Fast.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Normal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Aggressive.</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.10.2.5 Multithreading

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarThreads</td>
<td>Number of threads to use in the barrier solver</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 128}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CrossoverThreads</td>
<td>Number of threads to use in the crossover</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 128}</td>
<td></td>
</tr>
<tr>
<td>MipTasks</td>
<td>Number of MIP tasks in parallel (-1: automatic)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 256}</td>
<td></td>
</tr>
<tr>
<td>SimplexThreads</td>
<td>Number of threads to use in the simplex solver</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 128}</td>
<td></td>
</tr>
<tr>
<td>Threads</td>
<td>Number of threads to use</td>
<td>GAMS threads</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 128}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.10.2.6 Infeasibility Analysis

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| .feaspref | preference to include variable bound or equation side into feasibility relaxation  
The higher the value, the more likely it will be that associated variable bounds or equation sides are relaxed when computing a feasible relaxation. If zero, then the associated variable bounds or equation sides are not to considered for relaxation.  
Range: [0, ∞)                                                                 | 1.0 for equations, 0.0 for variables |
| FeasRelax | whether to compute a feasible relaxation instead of solving the problem  
Range: boolean  
Synonyms: feasopt                                                                 | 0                              |
| FeasRelaxMode | Specifies the feasibility relaxation mode  
0: Minimize sum of violations.  
1: Optimize original objective function under minimal sum of violations.  
2: Minimize number of violations.  
3: Optimize original objective function under minimal number of violations.  
4: Minimize sum of squared violations.  
5: Optimize original objective function under minimal sum of squared violations.  
Synonyms: feasoptmode                                                                 | 0                              |
| iis       | whether to compute an Irreducible Inconsistent Subsystem (IIS) if the problem is infeasible  
Range: boolean                                                                 | 0                              |
| IISMethod | Specifies the IIS method  
-1: Automatic.  
0: Find smaller IIS.  
1: Find IIS quickly.                                                                 | -1                             |

### 5.10.2.7 GAMS/COPT link

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>readparams</td>
<td>read COPT parameter file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
</tbody>
</table>
## 5.11 CPLEX

### 5.11.1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention. Access is provided (subject to proper licensing) to Cplex solution algorithms for linear, quadratically constrained and mixed integer programming problems. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

### 5.11.2 How to Run a Model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```plaintext
Option LP = Cplex;  { or QCP, MIP, MIQCP, RMIP or RMIQCP }
```

The above statement should appear before the Solve statement. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

Finally, a free of charge bare-bone interface to the LP and MIP solver of Cplex is available under the name **OSICPLEX**.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| solnpool    | Solution pool file name
              | The name of a solutions index gdx file for writing alternate solutions found by COPT. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions.
              | Range: string                                                              |         |
| solnpoolmerge| Solution pool file name for merged solutions                               |         |
| solvefinal  | whether to solve the LP obtained from fixing discrete variables and variables in SOS after a MIP solve
              | If marginals are not accessed after a MIP solve, it is advised to disable this option.
              | Range: boolean                                                             | 1       |
| writebas    | name of file to which to write advanced starting basis in basis format     |         |
| writebin    | name of file to which to write problem in COPT binary format               |         |
| writelp     | name of file to which to write problem in LP file format                   |         |
| writemps    | name of file to which to write problem in MPS file format                  |         |
| writemst    | name of file to which to write MIP starting point                          |         |
5.11.3 Overview of Cplex

5.11.3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex’s state of the art dual simplex algorithm. Certain types of problems benefit from using the primal simplex algorithm, the network optimizer, the barrier algorithm, or the sifting algorithm. The concurrent option will allow solving with different algorithms in parallel. The solution is returned by the first to finish.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled Physical Memory Limitations carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the primal simplex algorithm rather than the default dual simplex algorithm. Very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying primal simplex if numerical problems occur while using dual simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has > exactly 2 nonzero entries, one with a +1 > coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

Cplex provides a sifting algorithm which can be effective on problems with many more variables than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

GAMS/Cplex also provides access to the Cplex Conflict Refiner previously known as IIS (Irreducibly Inconsistent Set). The conflict refinder takes an infeasible program and produces a set of conflicting constraints. Such a set consists of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the conflict in terms of GAMS equation and variable names and includes the conflict report as part of the normal solution listing.

5.11.3.2 Quadratically Constrained Programming

Cplex can solve models with quadratic constraints. These are formulated in GAMS as models of type QCP. QCP models are solved with the Cplex Barrier method.

QP models are a special case that can be reformulated to have a quadratic objective function and only linear constraints. These are automatically reformulated from GAMS QCP models. When such problems are convex, Cplex normally solves them efficiently in polynomial time. Nonconvex QPs, however, are known to be quite hard. Cplex applies various approaches to those problems, such approaches as barrier algorithms or branch and bound algorithms. Notably, in the branch and bound approach, there is no theoretical guarantee about the complexity of such a problem. Consequently, solution of such a problem (that is, a nonconvex QP) can take many orders of magnitude longer than the solution of a convex QP of comparable dimensions. Therefore, the default is to reject such model. The parameter OptimalityTarget allows to change this behavior.

For QCP models Cplex returns a primal and in most cases also dual values.
5.11.3.3 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and cut algorithm which solves a series of LP subproblems. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

Cplex can also solve problems of GAMS model type MIQCP. As in the continuous case, if the base model is a QP the Simplex methods can be used and duals will be available at the solution. If the base model is a QCP, only the Barrier method can be used for the nodes and only primal values will be available at the solution.

5.11.3.4 Feasible Relaxation

The Conflict Refiner identifies the causes of infeasibility by means of inconsistent set of constraints. However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Cplex is known as FeasOpt (for Feasible Optimization) and turned on by parameter FeasOpt in a CPLEX option file. More details can be found in the section entitled Using the Feasibility Relaxation.

5.11.3.5 Solution Pool: Generating and Keeping Multiple Solutions

This section introduces the solution pool for storing multiple solutions to a mixed integer programming problem (MIP and MIQCP). The chapter also explains techniques for generating and managing those solutions.

The solution pool stores multiple solutions to a mixed integer programming (MIP and MIQCP) model. With this feature, you can direct the algorithm to generate multiple solutions in addition to the optimal solution. For example, some constraints may be difficult to formulate efficiently as linear expressions, or the objective may be difficult to quantify exactly. In such cases, obtaining multiple solutions will help you choose one which best fits all your criteria, including the criteria that could not be expressed easily in a conventional MIP or MIQCP model. For example,

- You can collect solutions within a given percentage of the optimal solution. To do so, apply the solution pool gap parameters SolnPoolAGap and SolnPoolGap.
- You can collect a set of diverse solutions. To do so, use the solution pool replacement parameter SolnPoolReplace to set the solution pool replacement strategy to 2. In order to control the diversity of solutions even more finely, apply a diversity filter.
- In an advanced application of this feature, you can collect solutions with specific properties. To do so, see the use of the incumbent filter.
- You can collect all solutions or all optimal solutions to model. To do so, set the solution pool intensity parameter SolnPoolIntensity to its highest value.

Please note, that the value for best possible can exceed the optimal solution value if CPLEX has already solved the model to optimality but continues to search for additional solutions.
5.11.3.5.1 Filling the Solution Pool  There are two ways to fill the solution pool associated with a model: You can accumulate successive incumbents or generate alternative solutions by populating the solution pool. The method is selected with the parameter \texttt{SolnPoolPop}:

- The regular optimization procedure automatically adds incumbents to the solution pool as they are discovered (\texttt{SolnPoolPop = 1}).
- Cplex also provides a procedure specifically to generate multiple solutions. You can invoke this procedure by setting option \texttt{SolnPoolPop = 2}. You can also invoke this procedure many times in a row in order to explore the solution space differently. In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory. This is done by specifying a GAMS program (option \texttt{SolnPoolPopRepeat}) that inspects the solutions. In case this GAMS program terminates normally, i.e. no execution or compilation error, the exploration for alternative solutions proceeds.

The option \texttt{SolnPoolReplace} designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity. The value 0 replaces solutions according to a first-in, first-out policy. The value 1 keeps the solutions with the best objective values. The value 2 replaces solutions in order to build a set of diverse solutions.

If the solutions you obtain are too similar to each other, try setting \texttt{SolnPoolReplace} to 2.

The replacement strategy applies only to the subset of solutions created in the current call of populate. Solutions already in the pool are not affected by the replacement strategy. They will not be replaced, even if they satisfy the criterion of the replacement strategy. So with every repeated call of the populate procedure the solution pool will be extended by the newly found solution. After the GAMS program specified in \texttt{SolnPoolPopRepeat} determined to continue the search for alternative solutions, the file specified by option \texttt{SolnPoolPopDel} option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

5.11.3.5.2 Enumerating All Solutions  With the solution pool, you can collect all solutions to a model. To do so, set the solution pool intensity parameter \texttt{SolnPoolIntensity} to its highest value, 4 and set \texttt{SolnPoolPop = 2}.

You can also enumerate all solutions that are valid for a specific criterion. For example, if you want to enumerate all alternative optimal solutions, do the following:

- Set the pool absolute gap parameter \texttt{SolnPoolAGap} = 0.0 (for models with potential numerical issues, better use a small non-zero like 1e-6).
- Set the pool intensity parameter \texttt{SolnPoolIntensity} = 4.
- Set the populate limit parameter \texttt{PopulateLim} to a value sufficiently large for your model; for example, 2100000000.
- Set the pool population parameter \texttt{SolnPoolPop} = 2.

Beware, however, that, even for small models, the number of possible solutions is likely to be huge. Consequently, enumerating all of them will take time and consume a large quantity of memory.

There may be an infinite number of possible values for a continuous variable, and it is not practical to enumerate all of them on a finite-precision computer. Therefore, populate gives only one solution for each set of binary and integer variables, even though there may exist several solutions that have the same values for all binary and integer variables but different values for continuous variables.

Likewise, for the same reason, the populate procedure does not generate all possible solutions for unbounded models. As soon as the proof of unboundedness is obtained, the populate procedure stops.

Cplex uses numerical methods of finite-precision arithmetic. Consequently, the feasibility of a solution depends on the value given to tolerances. Two parameters define the tolerances that assess the feasibility of a solution:
5.11 CPLEX

• the integrality tolerance EpInt
• the feasibility tolerance EpRHS

A solution may be considered feasible for one pair of values for these two parameters, and infeasible for a different pair. This phenomenon is especially noticeable in models with numeric difficulties, for example, in models with BigM coefficients.

Since the definition of a feasible solution is subject to tolerances, the total number of solutions to a model may vary, depending on the approach used to enumerate solutions, and on precisely which tolerances are used. In most models, this tolerance issue is not problematic. But, in the presence of numeric difficulties, Cplex may create solutions that are slightly infeasible or integer infeasible, and therefore create more solutions than expected.

5.11.3.5.3 Filtering the Solution Pool Filtering allows you to control properties of the solutions generated and stored in the solution pool. Cplex provides two predefined ways to filter solutions.

If you want to filter solutions based on their difference as compared to a reference solution, use a diversity filter. This filter is practical for most purposes. However, if you require finer control of which solutions to keep and which to eliminate, use the incumbent filter.

5.11.3.5.4 Diversity Filter A diversity filter allows you to generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables using dot option DivFlt and lower and upper bounds DivFltLo and DivFltUp. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution. If you need more than one diversity filter, for example, to generate solutions that share the characteristics of several different solutions, additional filters can be specified through a Cplex Filter File using parameter ReadFLT. Details can be found in the example model solnpool in the GAMS model library.

5.11.3.5.5 Incumbent Filter If you need to enforce more complex constraints on solutions (e.g. if you need to enforce nonlinear constraints), you can use the incumbent filtering. The incumbent checking routine is part of the GAMS BCH Facility. It will accept or reject incumbents independent of a solution pool. During the populate or regular optimize procedure, the incumbent checking routine specified by the parameter UserIncbCall is called each time a new solution is found, even if the new solution does not improve the objective value of the incumbent. The incumbent filter allows your application to accept or reject the new solution based on your own criteria. If the GAMS program specified by UserIncbCall terminates normally, the solution is rejected. If this program returns with a compilation or execution error, the incumbent is accepted.

5.11.3.5.6 Accessing the Solution Pool The solutions are stored in GAMS Data eXchange (GDX) file and can be loaded by your GAMS program. Details can be found in the model solnpool in the GAMS model library and in . If you instruct Cplex to generate thousands of solution this becomes inefficient. The option SolnPoolMerge triggers the creation of a single GDX file containing all solutions.

The GAMS/Cplex link produces, if properly instructed, a GAMS Data eXchange(GDX) file with name specified in SolnPool that contains a set Index with elements file1, file2, ... The associated text of these elements contain the file names of the individual GDX solution file. The name is constructed using the prefix soln (which can be specified differently by option SolnPoolPrefix), the name of the model and a sequence number. For example soln_loc_pl.gdx. GAMS/Cplex will overwrite existing GDX files without warning. The set Index allows us to conveniently walk through the different solutions in the solution pool. A complete model can be found in the model solnpool in the GAMS model library.
... solve mymodel min z using mip;
set soln possible solutions in the solution pool /file1*file1000/
   solnpool(soln) actual solutions;
file fsol;
execute_load 'solnpool.gdx', solnpool=Index;
loop(solnpool(soln),
   put_utility fsol 'gdxin' / solnpool.te(soln):0:0;
   execute_loadpoint;
   display z.l;
);

If you instruct Cplex to generate thousands of solution this method becomes inefficient. The option
SolnPooMerge triggers the creation of a single GDX file containing all solutions. Details on usage of this
option can be found in the model solmpool in the GAMS model library.

5.11.3.6 Benders Algorithm

CPLEX implements Benders algorithm.

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly
multiple) subproblems. To do so, CPLEX makes use of annotations that you supply for your model. This
approach can be applied to mixed-integer linear programs (MILP). For certain types of problems, this
approach can offer significant performance improvements.

The parameter, BendersStrategy, specifies to CPLEX how you want to apply Benders algorithm as a
strategy to solve your model. By default, if you did not annotate your model to specify a decomposition,
CPLEX executes conventional branch and bound. If you annotated your model, CPLEX attempts to
refine your decomposition and applies Benders algorithm. With this parameter, you can direct CPLEX to
decompose your model and to apply its implementation of Benders algorithm in one of these alternative
ways:

• 1: CPLEX attempts to decompose your model strictly according to your annotations.
• 2: CPLEX decomposes your model by using your annotations as hints and refining the decomposition
  where it can. CPLEX initially decomposes your model according to your annotation and then
  attempts to refine that decomposition by further decomposing the specified subproblems. This
  approach can be useful if you annotate certain variables to go into master, and all others to go into
  a single subproblem, which CPLEX can then decompose further for you.
• 3: CPLEX automatically decomposes your model, ignoring any annotations you may have supplied.
  CPLEX puts all integer variables into the master, puts all continuous variables into a subproblem
  and decomposes that subproblem, if possible.

If you want to specify a decomposition to CPLEX, you need to annotate your model and specify a Benders
partition of your variable space. These Benders partition can be conveniently specified with the dot option
BendersPartition or through the .stage variable suffix.

These Benders partition values specify to CPLEX whether certain variables belong to the master or to
one of the subproblems assigned to workers (where the subproblems are numbered from 1 (one) to N, the
number of subproblems). If you annotate a given variable with the value 0 (zero), CPLEX assigns that
variable to the master. If you annotate a given variable with the value i, where i is greater than or equal to
1 (one), CPLEX assigns that variable to subproblem i. If a variable is not specified, the default will be to
go into the master problem. Note that with variable.BendersPartition 1 you can assign all variables to the subproblem and then selectively assign the master variables with varname.BendersPartition 0.

If you want to communicate the Benders partition values via the .stage variable suffix, the partition numbers are off by one compared to the partition number via the dot option. So the master variables have stage 1, and the subproblems start with stage 2. If you want to leave a variable unassigned you can either make the stage 0 or fractional (e.g. 1.5). Discrete variables are automatically put into the master problem and don’t need to be set. In addition to specifying the Benders partition values via the .stage variable suffix the link option BendersPartitionInStage needs to be set to 1.

CPLEX produces an error if the annotated decomposition does not yield disjoint subproblems. For example, if your annotations specify that two (or more) variables belong in different subproblems, yet your model specifies that these variables participate in the same constraint, then these variables are linked. Consequently, the subproblems where these variables appear according to your annotations are not disjoint with respect to the partition.

5.11.3.7 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker’s shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Cplex provides tools that simplify the task: Cplex allows you to blend multiple objectives, to treat them hierarchically, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or lexicographic approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective, you only consider solutions that would not degrade the objective values of higher-priority objectives. Cplex allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Cplex will only solve multi-objective models with strictly linear objectives. Moreover, for continous models, Cplex will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:

```plaintext
equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defObj.. obj =e= numShifts - 1/100*sumPreferences;
defNumShifts.. numShifts =e= ....;
defSumPreferences.. sumPreferences =e= ....;

model workforce /all/;
solve workforce minimizing obj using mip;
```

With the default setting Cplex will solve the blended objective. Using the parameter MultObj Cplex will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (defObj). In the example, the numShifts objective with coefficient 1 has higher priority than the sumPreferences objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here numShifts will be minimized (same direction as on the solve statement) while sumPreferences will be maximized.
GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won't allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: \texttt{ObjNRelTol} and \texttt{ObjNAbsTol}. Their meaning differs for continuous and discrete models:

- **Discrete models**: By setting one of the two tolerances for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for \texttt{numShifts} is 100, and if we set \texttt{ObjNAbsTol} for this objective to 20, then the second optimization step maximizing \texttt{sumPreferences} would find the best solution for the second objective from among all solutions with objective 120 or better for \texttt{numShifts}. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).

- **Continuous Models**: \texttt{ObjNAbsTol} does not include a hard constraint for previous objective functions in subsequent solves. Instead, \texttt{ObjNAbsTol} defines a threshold for the reduced costs above which nonbasic variables in the associated LP solve will be fixed at the bound at which they reside. The option \texttt{ObjNRelTol} has no effect here. In order to apply the above discrete strategy for continuous models, set \texttt{MultiObjTolMip} to 1.

### 5.11.4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

- **Option BRatio = x;**
  
  Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

- **Option IterLim = n;**
  
  Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS.

  Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. The iteration limit is applied per node instead of as a total over all nodes. For MIP problems, controlling the length of the solution run by limiting the execution time (\texttt{ResLim}) is preferable.

  Similarly, when using the sifting algorithm, the iteration limit is applied per sifting iteration (i.e. per LP). The number of sifting iterations (LPs) can be limited by setting Cplex parameter \texttt{SiftItLim}. It is the number of sifting iterations that is reported back to GAMS as iterations used.

- **Option ResLim = x;**
  
  Sets the time limit in seconds. In case \texttt{ResLim} assumes its default value (1e+10) Cplex will use its own default (1e+75). The algorithm will terminate and pass on the current solution to GAMS.

- **Option SysOut = On;**
  
  Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.

- **\texttt{ModelName}.Cheat = x;**
Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option ObjDif overrides the GAMS cheat parameter.

- **ModelName.Cutoff = x;**
  Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.

- **ModelName.NodLim = x;**
  Maximum number of nodes to process for a MIP problem.

- **Option OptCA = x;**
  Absolute optimality criterion for a MIP problem. The OptCA option asks Cplex to stop when
  \[ |BP - BF| < OptCA \]
  where \( BF \) is the objective function value of the current best integer solution while \( BP \) is the best possible integer solution.
  Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

- **ModelName.OptCR = x;**
  Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when
  \[ \frac{|BP - BF|}{(1.0e - 10 + |BF|)} < OptCR \]
  where \( BF \) is the objective function value of the current best integer solution while \( BP \) is the best possible integer solution. The GAMS definition is:
  \[ \frac{|BP - BF|}{|BP|} < OptCR \]
  Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

- **ModelName.OptFile= 1;**
  Instructs Cplex to read the option file. The name of the option file is `cplex.opt`.

- **ModelName.PriorOpt= 1;**
  Instructs Cplex to use priority branching information passed by GAMS through the `variable.prior` parameters.

- **ModelName.TryInt= x;**
  Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. The preferred branching direction will only be effective when priorities are used. Priorities and tryint are sometimes not very effective and often outperformed by GAMS/CPLEX default settings. Supporting GAMS/CPLEX with knowledge about a known solution can be passed on by different means, please read more about this in section entitled Starting from a MIP Solution.

GAMS/Cplex also sets many model attributes that can by used in your GAMS program by accessing **ModelName.suffix**. A list of model attributes available after the solve can be found here Model Attributes Mainly Used After Solve.

Cplex has the concept of deterministic time i.e. a measure of time that respects the same solution path to arrive at the same values in the solution while it yields the same level of performance for repeated solving
of the same model with the same parameter settings, on the same computing platform. The length of
deterministic time tick may vary by platform. Nevertheless, ticks are normally consistent measures
for a given platform (combination of hardware and software) carrying the same load. In other words,
the correspondence of ticks to clock time depends on the hardware, software, and the current load of
the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant,
independent of the model solved. However, for very short optimization runs, the variation of this ratio
is typically high. GAMS/Cplex reports the deterministic time ticks spend inside Cplex in the model
attribute $\text{ETA1g}$. Normally $\text{ETA1g}$ is reporting the seconds.

5.11.5 Summary of CPLEX Options

The various Cplex options are listed here by category, with a few words about each to indicate its function.
The options are listed again, in alphabetical order and with detailed descriptions, in the last section of
this document.

5.11.5.1 Preprocessing and General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>advind</td>
<td>advanced basis use</td>
<td>determined by GAMS Bratio</td>
</tr>
<tr>
<td>aggfill</td>
<td>aggregator fill parameter</td>
<td>10</td>
</tr>
<tr>
<td>aggind</td>
<td>aggregator on/off</td>
<td>-1</td>
</tr>
<tr>
<td>bndrng</td>
<td>do lower / upper bound ranging</td>
<td></td>
</tr>
<tr>
<td>calcqcpsduals</td>
<td>calculate the dual values of a quadratically constrained problem</td>
<td>1</td>
</tr>
<tr>
<td>clocktype</td>
<td>clock type for computation time</td>
<td>2</td>
</tr>
<tr>
<td>coeredind</td>
<td>coefficient reduction on/off</td>
<td>-1</td>
</tr>
<tr>
<td>cpumask</td>
<td>switch and mask to bind threads to processors (Linux only)</td>
<td>auto</td>
</tr>
<tr>
<td>datacheck</td>
<td>controls data consistency checking and modeling assistance</td>
<td>0</td>
</tr>
<tr>
<td>depind</td>
<td>dependency checker on/off</td>
<td>-1</td>
</tr>
<tr>
<td>dettilim</td>
<td>deterministic time limit</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>feasopt</td>
<td>computes a minimum-cost relaxation to make an infeasible model feasible</td>
<td>0</td>
</tr>
<tr>
<td>feasoptmode</td>
<td>mode of FeasOpt</td>
<td>0</td>
</tr>
<tr>
<td>.feaspref</td>
<td>feasibility preference</td>
<td>1</td>
</tr>
<tr>
<td>fixoptfile</td>
<td>name of option file which is read just before solving the fixed problem</td>
<td></td>
</tr>
<tr>
<td>folding</td>
<td>LP folding will be attempted during the preprocessing phase</td>
<td>-1</td>
</tr>
<tr>
<td>freegamsmodel</td>
<td>preserves memory by dumping the GAMS model instance representation</td>
<td>0</td>
</tr>
<tr>
<td>iafie</td>
<td>secondary option file to be read in interactive mode triggered by iatriggerfile</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>iattriggerfile</td>
<td>file that triggers the reading of a secondary option file in interactive mode</td>
<td></td>
</tr>
<tr>
<td>iattriggertime</td>
<td>time interval in seconds the link looks for the trigger file in interactive mode</td>
<td>60</td>
</tr>
<tr>
<td>indicoptstrict</td>
<td>abort in case of an error in indicator constraint in solver option file</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>allow interactive option setting after a Control-C</td>
<td>0</td>
</tr>
<tr>
<td>lpmethod</td>
<td>algorithm to be used for LP problems</td>
<td>0</td>
</tr>
<tr>
<td>memoryemphasis</td>
<td>reduces use of memory</td>
<td>0</td>
</tr>
<tr>
<td>multobj</td>
<td>controls the hierarchical optimization of multiple objectives</td>
<td>0</td>
</tr>
<tr>
<td>multobjdisplay</td>
<td>level of display during multiobjective optimization</td>
<td>1</td>
</tr>
<tr>
<td>multobjmethod</td>
<td>method used for multi-objective solves</td>
<td>0</td>
</tr>
<tr>
<td>multobjoptfiles</td>
<td>List of option files used for individual solves within multi-objective optimisation</td>
<td></td>
</tr>
<tr>
<td>multobjtolmip</td>
<td>enables hard constraints for hierarchical optimization objectives based on degradation tolerances</td>
<td>1</td>
</tr>
<tr>
<td>names</td>
<td>load GAMS names into Cplex</td>
<td>1</td>
</tr>
<tr>
<td>numericalemphasis</td>
<td>emphasizes precision in numerically unstable or difficult problems</td>
<td>0</td>
</tr>
<tr>
<td>objnabstol</td>
<td>allowable absolute degradation for objective</td>
<td></td>
</tr>
<tr>
<td>objnreltol</td>
<td>allowable relative degradation for objective</td>
<td></td>
</tr>
<tr>
<td>objrng</td>
<td>do objective ranging</td>
<td>no objective ranging is done</td>
</tr>
<tr>
<td>optimalitytarget</td>
<td>type of optimality that Cplex targets</td>
<td>0</td>
</tr>
<tr>
<td>parallelmode</td>
<td>parallel optimization mode</td>
<td>1</td>
</tr>
<tr>
<td>predual</td>
<td>give dual problem to the optimizer</td>
<td>0</td>
</tr>
<tr>
<td>preind</td>
<td>turn presolver on/off</td>
<td>1</td>
</tr>
<tr>
<td>prepass</td>
<td>number of presolve applications to perform</td>
<td>-1</td>
</tr>
<tr>
<td>prereform</td>
<td>set presolve reformulations</td>
<td>3</td>
</tr>
<tr>
<td>printoptions</td>
<td>list values of all options to GAMS listing file</td>
<td>0</td>
</tr>
<tr>
<td>qextractalg</td>
<td>quadratic extraction algorithm in GAMS interface</td>
<td>0</td>
</tr>
<tr>
<td>qpmethod</td>
<td>algorithm to be used for QP problems</td>
<td>0</td>
</tr>
<tr>
<td>qtolin</td>
<td>linearization of the quadratic terms in the objective function of a QP or MIQP model</td>
<td>-1</td>
</tr>
<tr>
<td>randomseed</td>
<td>sets the random seed differently for diversity of solutions</td>
<td>changes with each Cplex release</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>readparams</td>
<td>read Cplex parameter file</td>
<td></td>
</tr>
<tr>
<td>reduce</td>
<td>primal and dual reduction type</td>
<td>3</td>
</tr>
<tr>
<td>relaxpreind</td>
<td>presolve for initial relaxation on/off</td>
<td>-1</td>
</tr>
<tr>
<td>rerun</td>
<td>rerun problem if presolve infeasible or unbounded</td>
<td>yes</td>
</tr>
<tr>
<td>rhsrng</td>
<td>do right-hand-side ranging</td>
<td>no right-hand-side ranging is done</td>
</tr>
<tr>
<td>rngrestart</td>
<td>write GAMS readable ranging information file</td>
<td>ranging information is printed to the listing file</td>
</tr>
<tr>
<td>scaind</td>
<td>matrix scaling on/off</td>
<td>0</td>
</tr>
<tr>
<td>solutiontype</td>
<td>type of solution (basic or non basic) for an LP or QP</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>global default thread count</td>
<td>GAMS Threads</td>
</tr>
<tr>
<td>tilim</td>
<td>overrides the GAMS ResLim option</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>tuning</td>
<td>invokes parameter tuning tool</td>
<td></td>
</tr>
<tr>
<td>tuningdettilim</td>
<td>tuning deterministic time limit per model or suite</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>tuningdisplay</td>
<td>level of information reported by the tuning tool</td>
<td>1</td>
</tr>
<tr>
<td>tuningmeasure</td>
<td>measure for evaluating progress for a suite of models</td>
<td>1</td>
</tr>
<tr>
<td>tuningrepeat</td>
<td>number of times tuning is to be repeated on perturbed versions</td>
<td>1</td>
</tr>
<tr>
<td>tuningtilim</td>
<td>tuning time limit per model or suite</td>
<td>0.2=GAMS ResLim</td>
</tr>
<tr>
<td>warninglimit</td>
<td>determines how many times warnings of a specific type (datacheck=2) will be displayed</td>
<td>10</td>
</tr>
<tr>
<td>workdir</td>
<td>directory for working files</td>
<td>current or project directory</td>
</tr>
<tr>
<td>workmem</td>
<td>memory available for working storage</td>
<td>2048.0</td>
</tr>
</tbody>
</table>

### 5.11.5.2 Simplex Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflictalg</td>
<td>algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model</td>
<td>0</td>
</tr>
<tr>
<td>conflictdisplay</td>
<td>decides how much information CPLEX reports when the conflict refiner is working</td>
<td>1</td>
</tr>
<tr>
<td>craind</td>
<td>crash strategy (used to obtain starting basis)</td>
<td>1</td>
</tr>
<tr>
<td>dprind</td>
<td>dual simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>dynamicrows</td>
<td>switch for dynamic management of rows</td>
<td>-1</td>
</tr>
<tr>
<td>epper</td>
<td>perturbation constant</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>iis</td>
<td>run the conflict refiner also known as IIS finder if the problem is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>netfind</td>
<td>attempt network extraction</td>
<td>2</td>
</tr>
<tr>
<td>netppriind</td>
<td>network simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>perind</td>
<td>force initial perturbation</td>
<td>0</td>
</tr>
<tr>
<td>perlim</td>
<td>number of stalled iterations before perturbation</td>
<td>0</td>
</tr>
<tr>
<td>ppriind</td>
<td>primal simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>pricelim</td>
<td>pricing candidate list</td>
<td>0, in which case it is determined automatically</td>
</tr>
<tr>
<td>reinv</td>
<td>refactorization frequency</td>
<td>0, in which case it is determined automatically</td>
</tr>
<tr>
<td>sifting</td>
<td>switch for sifting from simplex optimization</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.11.5.3 Simplex Limit Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>itlim</td>
<td>iteration limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>netitlim</td>
<td>iteration limit for network simplex</td>
<td>large</td>
</tr>
<tr>
<td>objllim</td>
<td>objective function lower limit</td>
<td>-1.0e+75</td>
</tr>
<tr>
<td>objulim</td>
<td>objective function upper limit</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>singlim</td>
<td>limit on singularity repairs</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.11.5.4 Simplex Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epmrk</td>
<td>Markowitz pivot tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>epopt</td>
<td>optimality tolerance</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>eprhs</td>
<td>feasibility tolerance</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>ltol</td>
<td>basis identification primal tolerance</td>
<td>0</td>
</tr>
<tr>
<td>mtol</td>
<td>basis identification dual tolerance</td>
<td>0</td>
</tr>
<tr>
<td>netepopt</td>
<td>optimality tolerance for the network simplex method</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>neteprhs</td>
<td>feasibility tolerance for the network simplex method</td>
<td>1.0e-06</td>
</tr>
</tbody>
</table>

### 5.11.5.5 Barrier Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baralg</td>
<td>algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barcolnz</td>
<td>dense column handling</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>barcrossalg</td>
<td>barrier crossover method</td>
<td>0</td>
</tr>
<tr>
<td>barepcomp</td>
<td>convergence tolerance</td>
<td>1.0e-08</td>
</tr>
<tr>
<td>bargrowth</td>
<td>unbounded face detection</td>
<td>1.0e+12</td>
</tr>
<tr>
<td>baritlim</td>
<td>iteration limit</td>
<td>large</td>
</tr>
<tr>
<td>barmaxcor</td>
<td>maximum correction limit</td>
<td>-1</td>
</tr>
<tr>
<td>barobjrng</td>
<td>maximum objective function</td>
<td>1.0e+20</td>
</tr>
<tr>
<td>barorder</td>
<td>row ordering algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barqcppepcomp</td>
<td>convergence tolerance for the barrier optimizer for QCPs</td>
<td>1.0e-07</td>
</tr>
<tr>
<td>barstartalg</td>
<td>barrier starting point algorithm</td>
<td>1</td>
</tr>
</tbody>
</table>

5.11.5.6 Sifting Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>siftalg</td>
<td>sifting subproblem algorithm</td>
<td>0</td>
</tr>
<tr>
<td>sftitlim</td>
<td>limit on sifting iterations</td>
<td>large</td>
</tr>
</tbody>
</table>

5.11.5.7 MIP Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbinterval</td>
<td>best bound interval</td>
<td>7</td>
</tr>
<tr>
<td>.benderspartition</td>
<td>Benders partition</td>
<td>0</td>
</tr>
<tr>
<td>benderspartitioninstage</td>
<td>Benders partition through stage variable suffix</td>
<td>0</td>
</tr>
<tr>
<td>bendersstrategy</td>
<td>Benders decomposition algorithm as a strategy</td>
<td>0</td>
</tr>
<tr>
<td>budstrenind</td>
<td>bound strengthening</td>
<td>-1</td>
</tr>
<tr>
<td>bqpcuts</td>
<td>boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality</td>
<td>0</td>
</tr>
<tr>
<td>brdir</td>
<td>set branching direction</td>
<td>0</td>
</tr>
<tr>
<td>bttol</td>
<td>backtracking limit</td>
<td>1.0</td>
</tr>
<tr>
<td>cliques</td>
<td>clique cut generation</td>
<td>0</td>
</tr>
<tr>
<td>covers</td>
<td>cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutlo</td>
<td>lower cutoff for tree search</td>
<td>-1.0e+75</td>
</tr>
<tr>
<td>cuts</td>
<td>default cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutsfactor</td>
<td>cut limit</td>
<td>-1.0</td>
</tr>
<tr>
<td>cutup</td>
<td>upper cutoff for tree search</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>disjcuts</td>
<td>disjunctive cuts generation</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>divetype</td>
<td>MIP dive strategy</td>
<td>0</td>
</tr>
<tr>
<td>eachcutlim</td>
<td>sets a limit for each type of cut</td>
<td>2100000000</td>
</tr>
<tr>
<td>flowcovers</td>
<td>flow cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>flowpaths</td>
<td>flow path cut generation</td>
<td>0</td>
</tr>
<tr>
<td>fpheur</td>
<td>feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>fraccuts</td>
<td>Gomory fractional cut generation</td>
<td>0</td>
</tr>
<tr>
<td>gubcovers</td>
<td>GUB cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>heurfreq</td>
<td>heuristic frequency</td>
<td>0</td>
</tr>
<tr>
<td>heuristic effort</td>
<td>the effort that CPLEX spends on heuristics during a MIP solve</td>
<td>1.0</td>
</tr>
<tr>
<td>implbd</td>
<td>implied bound cut generation</td>
<td>0</td>
</tr>
<tr>
<td>lazy</td>
<td>Lazy constraints activation</td>
<td>0</td>
</tr>
<tr>
<td>lazyconstraints</td>
<td>Indicator to use lazy constraints</td>
<td>0</td>
</tr>
<tr>
<td>lbheur</td>
<td>local branching heuristic</td>
<td>0</td>
</tr>
<tr>
<td>liftprojcuts</td>
<td>lift-and-project cuts</td>
<td>0</td>
</tr>
<tr>
<td>localimplied</td>
<td>generation of locally valid implied bound cuts</td>
<td>0</td>
</tr>
<tr>
<td>lowerobjstop</td>
<td>in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value</td>
<td>1e75</td>
</tr>
<tr>
<td>mcfcuts</td>
<td>multi-commodity flow cut generation</td>
<td>0</td>
</tr>
<tr>
<td>mipemphasis</td>
<td>MIP solution tactics</td>
<td>0</td>
</tr>
<tr>
<td>mipkappastats</td>
<td>MIP kappa computation</td>
<td>0</td>
</tr>
<tr>
<td>mipordind</td>
<td>priority list on/off</td>
<td>GAMS PriorOpt</td>
</tr>
<tr>
<td>mipordtype</td>
<td>priority order generation</td>
<td>0</td>
</tr>
<tr>
<td>mipsearch</td>
<td>search strategy for mixed integer programs</td>
<td>0</td>
</tr>
<tr>
<td>mipstart</td>
<td>use mip starting values</td>
<td>0</td>
</tr>
<tr>
<td>mipstopexpr</td>
<td>stopping expression for branch and bound</td>
<td>0</td>
</tr>
<tr>
<td>miqcpstrat</td>
<td>MIQCP relaxation choice</td>
<td>0</td>
</tr>
<tr>
<td>mircuts</td>
<td>mixed integer rounding cut generation</td>
<td>0</td>
</tr>
<tr>
<td>multin mipstart</td>
<td>use multiple mipstarts provided via gdx files</td>
<td>0</td>
</tr>
<tr>
<td>nodecuts</td>
<td>decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree</td>
<td>0</td>
</tr>
<tr>
<td>nodefileind</td>
<td>node storage file indicator</td>
<td>1</td>
</tr>
<tr>
<td>nodesel</td>
<td>node selection strategy</td>
<td>1</td>
</tr>
<tr>
<td>preslvnd</td>
<td>node presolve selector</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>probe</td>
<td>perform probing before solving a MIP</td>
<td>0</td>
</tr>
<tr>
<td>qpmakepsdind</td>
<td>adjust MIQP formulation to make the quadratic matrix positive-semi-definite</td>
<td>1</td>
</tr>
<tr>
<td>relaxfixedinfeas</td>
<td>accept small infeasibilities in the solve of the fixed problem</td>
<td>0</td>
</tr>
<tr>
<td>repeatpresolve</td>
<td>reapply presolve at root after preprocessing</td>
<td>-1</td>
</tr>
<tr>
<td>rinsheur</td>
<td>relaxation induced neighborhood search frequency</td>
<td>0</td>
</tr>
<tr>
<td>rltcuts</td>
<td>Reformulation Linearization Technique (RLT) cuts</td>
<td>0</td>
</tr>
<tr>
<td>solvefinal</td>
<td>switch to solve the problem with fixed discrete variables</td>
<td>1</td>
</tr>
<tr>
<td>sos1reform</td>
<td>automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)</td>
<td>0</td>
</tr>
<tr>
<td>sos2reform</td>
<td>automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)</td>
<td>0</td>
</tr>
<tr>
<td>startalg</td>
<td>MIP starting algorithm</td>
<td>0</td>
</tr>
<tr>
<td>strongcandlim</td>
<td>size of the candidates list for strong branching</td>
<td>10</td>
</tr>
<tr>
<td>strongitlim</td>
<td>limit on iterations per branch for strong branching</td>
<td>0</td>
</tr>
<tr>
<td>subalg</td>
<td>algorithm for subproblems</td>
<td>0</td>
</tr>
<tr>
<td>submipnodelim</td>
<td>limit on number of nodes in an RINS subMIP</td>
<td>500</td>
</tr>
<tr>
<td>submipscale</td>
<td>scale the problem matrix when CPLEX solves a subMIP during MIP optimization</td>
<td>0</td>
</tr>
<tr>
<td>submipstartalg</td>
<td>starting algorithm for a subMIP of a MIP</td>
<td>0</td>
</tr>
<tr>
<td>submipsubalg</td>
<td>algorithm for subproblems of a subMIP of a MIP</td>
<td>0</td>
</tr>
<tr>
<td>symmetry</td>
<td>symmetry breaking cuts</td>
<td>-1</td>
</tr>
<tr>
<td>upperobjstop</td>
<td>in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value</td>
<td>$1e75$</td>
</tr>
<tr>
<td>.usercut</td>
<td>User cut activation</td>
<td>0</td>
</tr>
<tr>
<td>.usercutpool</td>
<td>Indicator to use user cuts</td>
<td>0</td>
</tr>
<tr>
<td>varsel</td>
<td>variable selection strategy at each node</td>
<td>0</td>
</tr>
<tr>
<td>workeralgorithm</td>
<td>set method for optimizing benders subproblems</td>
<td>0</td>
</tr>
<tr>
<td>zerohalfcuts</td>
<td>zero-half cuts</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.11.5.8 MIP Limit Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggcutlim</td>
<td>aggregation limit for cut generation</td>
<td>3</td>
</tr>
<tr>
<td>auxrootthreads</td>
<td>number of threads for auxiliary tasks at the root node</td>
<td>0</td>
</tr>
<tr>
<td>cutpass</td>
<td>maximum number of cutting plane passes</td>
<td>0</td>
</tr>
<tr>
<td>fraccand</td>
<td>candidate limit for generating Gomory fractional cuts</td>
<td>200</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>fracpass</td>
<td>maximum number of passes for generating Gomory fractional cuts</td>
<td>0</td>
</tr>
<tr>
<td>intsollim</td>
<td>maximum number of integer solutions</td>
<td>large</td>
</tr>
<tr>
<td>nodelim</td>
<td>maximum number of nodes to solve</td>
<td>GAMS NodLim</td>
</tr>
<tr>
<td>polishafterdettim</td>
<td>deterministic time before starting to polish a feasible solution</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>polishafterepagap</td>
<td>absolute MIP gap before starting to polish a feasible solution</td>
<td>0.0</td>
</tr>
<tr>
<td>polishafterepgap</td>
<td>relative MIP gap before starting to polish a solution</td>
<td>0.0</td>
</tr>
<tr>
<td>polishafterintsol</td>
<td>MIP integer solutions to find before starting to polish a feasible solution</td>
<td>2147483647</td>
</tr>
<tr>
<td>polishafternode</td>
<td>nodes to process before starting to polish a feasible solution</td>
<td>2147483647</td>
</tr>
<tr>
<td>polishaftertime</td>
<td>time before starting to polish a feasible solution</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>probeletttime</td>
<td>deterministic time spent probing</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>probetime</td>
<td>time spent probing</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>repairtries</td>
<td>try to repair infeasible MIP start</td>
<td>0</td>
</tr>
<tr>
<td>trelim</td>
<td>maximum space in memory for tree</td>
<td>1.0e+75</td>
</tr>
</tbody>
</table>

### 5.11.5.9 MIP Solution Pool Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>.divflt</td>
<td>solution pool range filter coefficients</td>
<td>0</td>
</tr>
<tr>
<td>divfltllo</td>
<td>lower bound on diversity</td>
<td>mindouble</td>
</tr>
<tr>
<td>divfltup</td>
<td>upper bound on diversity</td>
<td>maxdouble</td>
</tr>
<tr>
<td>populatelim</td>
<td>limit of solutions generated for the solution pool by populate method</td>
<td>20</td>
</tr>
<tr>
<td>readflt</td>
<td>reads Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td>solnpool</td>
<td>solution pool file name</td>
<td></td>
</tr>
<tr>
<td>solnpoolagap</td>
<td>absolute tolerance for the solutions in the solution pool</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>solnpoolcapacity</td>
<td>limits of solutions kept in the solution pool</td>
<td>21000000000</td>
</tr>
<tr>
<td>solnpoolgap</td>
<td>relative tolerance for the solutions in the solution pool</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>solnpoolintensity</td>
<td>solution pool intensity for ability to produce multiple solutions</td>
<td>0</td>
</tr>
<tr>
<td>solnpoolmerge</td>
<td>solution pool file name for merged solutions</td>
<td></td>
</tr>
<tr>
<td>solnpoolnumsym</td>
<td>maximum number of variable symbols when writing merged solutions</td>
<td>10</td>
</tr>
<tr>
<td>solnpoolpop</td>
<td>methods to populate the solution pool</td>
<td>1</td>
</tr>
<tr>
<td>solnpoolpopdel</td>
<td>file with solution numbers to delete from the solution pool</td>
<td></td>
</tr>
<tr>
<td>solnpoolpoprepeat</td>
<td>method to decide if populating the solution should be repeated</td>
<td></td>
</tr>
<tr>
<td>solnpoolprefix</td>
<td>file name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>solnpoolreplace</td>
<td>strategy for replacing a solution in the solution pool</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.11.5.10 MIP Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bendersfeascuttol</td>
<td>Tolerance for whether a feasibility cut has been violated in Benders decomposition</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>bendersoptcuttol</td>
<td>Tolerance for optimality cuts in Benders decomposition</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>epagap</td>
<td>absolute stopping tolerance</td>
<td></td>
</tr>
<tr>
<td>epgap</td>
<td>relative stopping tolerance</td>
<td></td>
</tr>
<tr>
<td>epint</td>
<td>integrality tolerance</td>
<td>1.0e-05</td>
</tr>
<tr>
<td>eplin</td>
<td>degree of tolerance used in linearization</td>
<td>0.001</td>
</tr>
<tr>
<td>objdif</td>
<td>overrides GAMS Cheat parameter</td>
<td>0.0</td>
</tr>
<tr>
<td>relobjdiff</td>
<td>relative cheat parameter</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### 5.11.5.11 Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bardisplay</td>
<td>progress display level</td>
<td>1</td>
</tr>
<tr>
<td>clonelog</td>
<td>enable clone logs</td>
<td>0</td>
</tr>
<tr>
<td>exactkappa</td>
<td>report exact condition number in quality report</td>
<td>0</td>
</tr>
<tr>
<td>mipdisplay</td>
<td>progress display level</td>
<td>4</td>
</tr>
<tr>
<td>mipinterval</td>
<td>progress display interval</td>
<td>0</td>
</tr>
<tr>
<td>miptrace</td>
<td>filename of MIP trace file</td>
<td></td>
</tr>
<tr>
<td>mpslongnum</td>
<td>MPS file format precision of numeric output</td>
<td>1</td>
</tr>
<tr>
<td>netdisplay</td>
<td>network display level</td>
<td>2</td>
</tr>
<tr>
<td>quality</td>
<td>write solution quality statistics</td>
<td>0</td>
</tr>
<tr>
<td>siftdisplay</td>
<td>sifting display level</td>
<td>1</td>
</tr>
<tr>
<td>simdisplay</td>
<td>simplex display level</td>
<td>1</td>
</tr>
<tr>
<td>writeannotation</td>
<td>produce a Cplex annotation file</td>
<td></td>
</tr>
<tr>
<td>writebas</td>
<td>produce a Cplex basis file</td>
<td></td>
</tr>
<tr>
<td>writeflt</td>
<td>produce a Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td>writelp</td>
<td>produce a Cplex LP file</td>
<td></td>
</tr>
<tr>
<td>writemst</td>
<td>produce a Cplex mst file</td>
<td></td>
</tr>
</tbody>
</table>
### 5.11 CPLEX

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>writeord</strong></td>
<td>produce a Cplex ord file</td>
<td></td>
</tr>
<tr>
<td><strong>writeparam</strong></td>
<td>produce a Cplex parameter file with all active options</td>
<td></td>
</tr>
<tr>
<td><strong>writepre</strong></td>
<td>produce a Cplex LP/MPS/SAV file of the presolved problem</td>
<td></td>
</tr>
<tr>
<td><strong>writeprob</strong></td>
<td>produce a Cplex problem file and inferrs the type from the extension</td>
<td></td>
</tr>
<tr>
<td><strong>writesav</strong></td>
<td>produce a Cplex binary problem file</td>
<td></td>
</tr>
</tbody>
</table>

### 5.11.5.12 BCH Facility Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>usercallparmfile</strong></td>
<td>Command-line parameter include file used in GAMS command-line calls triggered by BCH</td>
<td></td>
</tr>
<tr>
<td><strong>usercutcall</strong></td>
<td>the GAMS command line to call the cut generator</td>
<td></td>
</tr>
<tr>
<td><strong>usercutfirst</strong></td>
<td>calls the cut generator for the first n nodes</td>
<td>10</td>
</tr>
<tr>
<td><strong>usercutfreq</strong></td>
<td>determines the frequency of the cut generator model calls</td>
<td>10</td>
</tr>
<tr>
<td><strong>usercutinterval</strong></td>
<td>determines the interval when to apply the multiplier for the frequency of the cut generator model calls</td>
<td>100</td>
</tr>
<tr>
<td><strong>usercutmult</strong></td>
<td>determines the multiplier for the frequency of the cut generator model calls</td>
<td>2</td>
</tr>
<tr>
<td><strong>usercutnewint</strong></td>
<td>calls the cut generator if the solver found a new integer feasible solution</td>
<td>1</td>
</tr>
<tr>
<td><strong>usergdxin</strong></td>
<td>the name of the GDX file read back into Cplex</td>
<td>bchin.gdx</td>
</tr>
<tr>
<td><strong>usergdxname</strong></td>
<td>the name of the GDX file exported from the solver with the solution at the node</td>
<td>bchout.gdx</td>
</tr>
<tr>
<td><strong>usergdxnameinc</strong></td>
<td>the name of the GDX file exported from the solver with the incumbent solution</td>
<td>bchout_i.gdx</td>
</tr>
<tr>
<td><strong>usergdxprefix</strong></td>
<td>prefixes usergdxin, usergdxname, and usergdxnameinc</td>
<td></td>
</tr>
<tr>
<td><strong>usergdxsol</strong></td>
<td>the name of the GDX file exported by Cplex to store the solution of extra columns</td>
<td>bchsol.gdx</td>
</tr>
<tr>
<td><strong>userheurcall</strong></td>
<td>the GAMS command line to call the heuristic</td>
<td></td>
</tr>
<tr>
<td><strong>userheurfirst</strong></td>
<td>calls the heuristic for the first n nodes</td>
<td>10</td>
</tr>
<tr>
<td><strong>userheurfreq</strong></td>
<td>determines the frequency of the heuristic model calls</td>
<td>10</td>
</tr>
<tr>
<td><strong>userheurinterval</strong></td>
<td>determines the interval when to apply the multiplier for the frequency of the heuristic model calls</td>
<td>100</td>
</tr>
<tr>
<td><strong>userheurmult</strong></td>
<td>determines the multiplier for the frequency of the heuristic model calls</td>
<td>2</td>
</tr>
<tr>
<td><strong>userheurnewint</strong></td>
<td>calls the heuristic if the solver found a new integer feasible solution</td>
<td>1</td>
</tr>
<tr>
<td><strong>userheurobjfirst</strong></td>
<td>Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement</td>
<td>0</td>
</tr>
<tr>
<td><strong>userincbcall</strong></td>
<td>the GAMS command line to call the incumbent checking program</td>
<td></td>
</tr>
<tr>
<td><strong>userincbicall</strong></td>
<td>the GAMS command line to call the incumbent reporting program</td>
<td></td>
</tr>
<tr>
<td><strong>userjobid</strong></td>
<td>postfixes if, o on call adds –userjobid to the call. Postfixes gdxname, gdxnameinc and gdxin</td>
<td></td>
</tr>
</tbody>
</table>
### 5.11.5.13 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (∗) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file `cplex.opt`.

```plaintext
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

### 5.11.6 Special Notes

#### 5.11.6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically. Also consider option `MemoryEmphasis` to conserve memory where possible.

When solving LPs, the Simplex algorithm (lpmethod 1 or 2) usually consumes less memory than the Barrier algorithm (lpmethod 2). During the Simplex algorithm, Cplex performs an operation called refactorization at a frequency determined by the `ReInv` option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorization is an expensive operation, increasing the refactorization frequency by reducing the `ReInv` option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

GAMS/Cplex also provides the option `FreeGAMSModel` to free some memory allocated by the GAMS link and making it available to Cplex. This only works when the GAMS parameter `SolveLink` is set 0 which should be always done when memory is tight because GAMS completely vacates memory.

The `Threads` options also has a significant impact on memory consumption. The concurrent solvers keep multiple copies of the problem in memory which double or even triples the amount of memory consumed. If memory tight, set the `Threads` parameter to 1.

Not loading GAMS names into CPLEX can also help in saving some memory. This can be done by setting the model attribute `dictFile` or by using CPLEX option `names`.

---

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>userkeep</td>
<td>calls gamskeep instead of gams</td>
<td>0</td>
</tr>
<tr>
<td>userlazyconcall</td>
<td>the GAMS command line to call the lazy constraint generator</td>
<td></td>
</tr>
</tbody>
</table>
5.11.6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare SOS1 and SOS2 variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

- A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

- A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.

5.11.6.3 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of semi-continuous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```plaintext
SemiCont Variable x;
x.lo = 3.2;
x.up = 8.7;

SemiInt Variable y;
y.lo = 5;
y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

5.11.6.4 Running Out of Memory for MIP Problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limited by using the WorkMem, option with the NodeFileInd option. Setting NodeFileInd to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option WorkMem. That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.
Set option NodeSel to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.

Set option VarSel to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.

On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using option Cuts.

Use the Simplex algorithm instead of the Barrier algorithm using options StartAlg and SubAlg.

Use option solvefinal to avoid solving the final LP.

See also Physical Memory Limitations.

### 5.11.6.5 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as $2^n$ nodes, where $n$ equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer's memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using some different option settings.

### 5.11.6.6 Starting from a MIP Solution

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution. When you provide such a starting solution, you may invoke relaxation induced neighborhood search (RINS heuristic) or solution polishing to improve the given solution. This first integer solution may include continuous and discrete variables of various types, such as semi-continuous variables or special ordered sets.

If you specify values for all discrete variables, GAMS/CPLEX will check the validity of the values as an integer-feasible solution; if you specify values for only a portion of the discrete variables, GAMS/CPLEX will attempt to fill in the missing values in a way that leads to an integer-feasible solution. If the specified values do not lead directly to an integer-feasible solution, GAMS/CPLEX will apply a quick heuristic to try to repair the MIP Start. The number of times that GAMS/CPLEX applies the heuristic is controlled by the repair tries parameter (RepairTries). If this process succeeds, the solution will be treated as an integer solution of the current problem.

A MIP start will only be used by GAMS/CPLEX if the MipStart parameter is set to 1. The starting values must be set via the .L variable attribute in the GAMS model before the solve statement.
5.11.6.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasOpt parameter in a CPLEX solver option file.

With the FeasOpt option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the dot option FeasPref. A negative or zero preference means that the associated bound or constraint is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a CPLEX solver option file. The syntax is:

(variable or equation) .feaspref (value)

For example, suppose we have a GAMS declaration:

Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);

Then, the relaxation preference in the cplex.opt file can be specified by:

```
feasopt 1
v.feaspref 1
v.feaspref('i1','*) 2
v.feaspref('i1','j2') 0
e.feaspref('','j1') 0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables v(i,j) have preference of 1, except variables over set element i1, which have a preference of 2. The variable over set element i1 and j2 has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords variables or equations instead of the individual variable and equations names (e.g. variables.feaspref 1).

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations). Please check description of parameters FeasOpt and FeasOptMode for details. Also check example models feasopt* in the GAMS Model library.
5.11.6.8 Sensitivity Analysis

Sensitivity analysis (post-optimality analysis) in linear programming allows one to find out more about an optimal solution for a problem. In particular, objective ranging and right-hand-side ranging give information about how much an objective coefficient or a right-hand-side value can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or a right-hand-side.

Although not so much used for practical large scale problems and not available for mixed-integer or nonlinear models, ranging information can still be of use in some circumstances. This section describes how to produce ranging information with GAMS/CPLEX.

To obtain objective ranging information for a particular variable, the name of the GAMS variable should be specified with the ObjRng option. For example, to obtain ranging information for a variable prod, add the line

```
objrng prod
```
to a CPLEX options file. The ObjRng option can be repeated to specify ranging for more than one variable. To specify ranging for all variables, use the keyword all, i.e.,

```
objrng all
```

Similarly, to obtain right-hand-side ranging information for a particular equation, the name of the equation should be specified with the RhsRng option. Also this option can be repeated to obtain right-hand-side ranging information for several equations. To specify ranging for all equations use the keyword all.

As an example, consider solving the model [TRNSPORT] from the GAMS model library with CPLEX and options file

```
objrng all
rhrng all
```

This gives the following table in the listing file:

```
<table>
<thead>
<tr>
<th>EQUATION NAME</th>
<th>LOWER</th>
<th>CURRENT</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>-INF</td>
<td>0</td>
<td>+INF</td>
</tr>
<tr>
<td>supply(seattle)</td>
<td>300</td>
<td>350</td>
<td>625</td>
</tr>
<tr>
<td>supply(san-diego)</td>
<td>550</td>
<td>600</td>
<td>+INF</td>
</tr>
<tr>
<td>demand(new-york)</td>
<td>50</td>
<td>325</td>
<td>375</td>
</tr>
<tr>
<td>demand(chicago)</td>
<td>25</td>
<td>300</td>
<td>350</td>
</tr>
<tr>
<td>demand(topeka)</td>
<td>0</td>
<td>275</td>
<td>325</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VARIABLE NAME</th>
<th>LOWER</th>
<th>CURRENT</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(seattle, new-york)</td>
<td>0.216</td>
<td>0.225</td>
<td>0.225</td>
</tr>
<tr>
<td>x(seattle, chicago)</td>
<td>0</td>
<td>0.153</td>
<td>0.162</td>
</tr>
<tr>
<td>x(seattle, topeka)</td>
<td>0.126</td>
<td>0.162</td>
<td>+INF</td>
</tr>
<tr>
<td>x(san-diego, new-york)</td>
<td>0.225</td>
<td>0.225</td>
<td>0.234</td>
</tr>
<tr>
<td>x(san-diego, chicago)</td>
<td>0.153</td>
<td>0.162</td>
<td>+INF</td>
</tr>
<tr>
<td>x(san-diego, topeka)</td>
<td>0</td>
<td>0.126</td>
<td>0.162</td>
</tr>
<tr>
<td>z</td>
<td>-INF</td>
<td>1</td>
<td>+INF</td>
</tr>
</tbody>
</table>
```
If obtaining ranging information in a listing file is not sufficient, option \texttt{RngRestart} can be used to specify a file to which to write ranging information in GAMS syntax. For example, using an options file containing

\begin{verbatim}
  rhsrng supply
  rhsrng demand
  rngrestart ranges.inc
\end{verbatim}

will result in a file named \texttt{ranges.inc} being written with the following content:

\begin{verbatim}
* Include file with ranging information
* The set RNGLIM /LO,UP/ is assumed to be declared.
PARAMETER supplyRNG(i,RNGLIM) /
  seattle.LO  300
  seattle.UP  625
  san-diego.LO 550
  san-diego.UP +INF
/;
PARAMETER demandRNG(j,RNGLIM) /
  new-york.LO   50
  new-york.UP  375
  chicago.LO   25
  chicago.UP   350
  topeka.LO    0
  topeka.UP   325
/;
\end{verbatim}

For each equation specified, the ranging information is stored in a newly declared corresponding GAMS parameter. The name of the parameter is based on the name of the equation, but with \texttt{RNG} appended. The user is responsible for ensuring that the new name does not exceed the maximum symbol name length of GAMS identifiers. Further, the domain list of the new parameter is the same as the domain list for the corresponding equation with an additional dimension added to the end. The user is responsible for ensuring that the new parameter does not exceed the maximum number of index positions.

5.11.7 GAMS/Cplex Log File

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option \texttt{simdisplay} there is a log line for each refactorization. The screen log has the following appearance:

\begin{verbatim}
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.01 sec.
Using conservative initial basis.
\end{verbatim}
Iteration log . . .
Iteration:  1  Scaled infeas  =  193998.067174
Iteration:  29  Objective  =  -3484.286415
Switched to devex.
Iteration:  98  Objective  =  -1852.931117
Iteration:  166  Objective  =  -349.706562

Optimal solution found.

Objective :  901.161538

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =  0.01 sec.

Iteration log . . .
Iteration:  1  Scaled dual infeas  =  3.890823
Iteration:  53  Dual objective  =  4844.392441
Iteration:  114  Dual objective  =  1794.360714
Iteration:  176  Dual objective  =  1120.183325
Iteration:  238  Dual objective  =  915.143030
Removing shift (1).

Optimal solution found.

Objective :  901.161538

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =  0.01 sec.

Iteration log . . .
Iteration:  0  Infeasibility  =  1232.378800 (-1.32326e+12)

Network - Optimal:  Objective  =  1.5716820779e+03
Network time =  0.01 sec.  Iterations = 26 (24)

Iteration log . . .
Iteration:  1  Scaled infeas  =  212696.154729
Iteration:  62  Scaled infeas  =  10020.401232
Iteration:  142  Scaled infeas  =  4985.200129
Switched to devex.
Iteration:  217  Objective  =  -3883.782587
Iteration: 291  Objective  =  -1423.126582

Optimal solution found.

Objective : 901.161538

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.02 sec.
Number of nonzeros in lower triangle of A'A' = 6545
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.01 sec.

Summary statistics for Cholesky factor:

<table>
<thead>
<tr>
<th>Rows in Factor</th>
<th>Integer space required</th>
<th>Total non-zeros in factor</th>
<th>Total FP ops to factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>243</td>
<td>578</td>
<td>8491</td>
<td>410889</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itn</th>
<th>Primal Obj</th>
<th>Dual Obj</th>
<th>Prim Inf</th>
<th>Upper Inf</th>
<th>Dual Inf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.2826603e+06</td>
<td>7.4700787e+08</td>
<td>2.25e+10</td>
<td>6.13e+06</td>
<td>4.00e+05</td>
</tr>
<tr>
<td>1</td>
<td>-2.6426195e+05</td>
<td>6.3852653e+08</td>
<td>4.58e+09</td>
<td>1.25e+06</td>
<td>1.35e+05</td>
</tr>
<tr>
<td>2</td>
<td>-9.917854e+04</td>
<td>4.1669756e+08</td>
<td>1.66e+09</td>
<td>4.52e+05</td>
<td>3.93e+04</td>
</tr>
<tr>
<td>3</td>
<td>-2.6624468e+04</td>
<td>2.1507018e+08</td>
<td>3.80e+08</td>
<td>1.04e+05</td>
<td>1.20e+04</td>
</tr>
<tr>
<td>4</td>
<td>-1.2104334e+04</td>
<td>7.8532364e+07</td>
<td>9.69e+07</td>
<td>2.65e+04</td>
<td>2.52e+03</td>
</tr>
<tr>
<td>5</td>
<td>-9.5217661e+03</td>
<td>4.2663811e+07</td>
<td>2.81e+07</td>
<td>7.67e+03</td>
<td>9.92e+02</td>
</tr>
<tr>
<td>6</td>
<td>-8.6929410e+03</td>
<td>1.4134077e+07</td>
<td>4.94e+06</td>
<td>1.35e+03</td>
<td>2.16e+02</td>
</tr>
<tr>
<td>7</td>
<td>-8.3726267e+03</td>
<td>3.1619431e+06</td>
<td>3.13e-07</td>
<td>6.84e-12</td>
<td>3.72e+01</td>
</tr>
<tr>
<td>8</td>
<td>-8.2862559e+03</td>
<td>3.3985844e+03</td>
<td>1.43e-08</td>
<td>5.60e-12</td>
<td>3.98e-02</td>
</tr>
<tr>
<td>9</td>
<td>-3.8181279e+03</td>
<td>2.6166059e+03</td>
<td>1.58e-08</td>
<td>9.37e-12</td>
<td>2.50e-02</td>
</tr>
<tr>
<td>10</td>
<td>-5.1366439e+03</td>
<td>2.8102021e+03</td>
<td>3.90e-06</td>
<td>7.34e-12</td>
<td>1.78e-02</td>
</tr>
<tr>
<td>11</td>
<td>-1.9771576e+03</td>
<td>1.5960442e+03</td>
<td>3.43e-06</td>
<td>7.02e-12</td>
<td>3.81e-03</td>
</tr>
<tr>
<td>12</td>
<td>-4.3346261e+02</td>
<td>8.3443795e+02</td>
<td>4.99e-07</td>
<td>1.22e-11</td>
<td>7.93e-04</td>
</tr>
<tr>
<td>13</td>
<td>1.882968e+02</td>
<td>5.2138155e+02</td>
<td>2.22e-07</td>
<td>1.45e-11</td>
<td>8.72e-04</td>
</tr>
<tr>
<td>14</td>
<td>5.0418542e+02</td>
<td>5.3676806e+02</td>
<td>1.46e-07</td>
<td>1.26e-11</td>
<td>7.93e-04</td>
</tr>
<tr>
<td>15</td>
<td>2.4951043e+02</td>
<td>6.5911879e+02</td>
<td>1.73e-07</td>
<td>1.43e-11</td>
<td>5.33e-04</td>
</tr>
<tr>
<td>16</td>
<td>2.4666057e+02</td>
<td>7.6179064e+02</td>
<td>7.83e-06</td>
<td>2.17e-11</td>
<td>3.15e-04</td>
</tr>
<tr>
<td>17</td>
<td>4.6820025e+02</td>
<td>8.1319322e+02</td>
<td>4.75e-06</td>
<td>1.78e-11</td>
<td>2.57e-04</td>
</tr>
<tr>
<td>18</td>
<td>5.6081604e+02</td>
<td>7.9608915e+02</td>
<td>3.09e-06</td>
<td>1.98e-11</td>
<td>2.89e-04</td>
</tr>
<tr>
<td>19</td>
<td>6.4517294e+02</td>
<td>7.7729659e+02</td>
<td>1.61e-06</td>
<td>1.27e-11</td>
<td>3.29e-04</td>
</tr>
<tr>
<td>20</td>
<td>7.9603053e+02</td>
<td>7.8584631e+02</td>
<td>5.91e-07</td>
<td>1.91e-11</td>
<td>3.00e-04</td>
</tr>
<tr>
<td>21</td>
<td>8.5871436e+02</td>
<td>8.0198336e+02</td>
<td>1.32e-07</td>
<td>1.46e-11</td>
<td>2.57e-04</td>
</tr>
<tr>
<td>22</td>
<td>8.8146686e+02</td>
<td>8.1244367e+02</td>
<td>1.46e-07</td>
<td>1.84e-11</td>
<td>2.29e-04</td>
</tr>
<tr>
<td>23</td>
<td>8.8327998e+02</td>
<td>8.3545696e+02</td>
<td>1.44e-07</td>
<td>1.96e-11</td>
<td>1.71e-04</td>
</tr>
<tr>
<td>24</td>
<td>8.8595062e+02</td>
<td>8.4926550e+02</td>
<td>1.30e-07</td>
<td>2.85e-11</td>
<td>1.35e-04</td>
</tr>
<tr>
<td>25</td>
<td>8.9780584e+02</td>
<td>8.6318712e+02</td>
<td>1.60e-07</td>
<td>1.08e-11</td>
<td>9.89e-05</td>
</tr>
<tr>
<td>26</td>
<td>8.9940069e+02</td>
<td>8.9108502e+02</td>
<td>1.78e-07</td>
<td>1.07e-11</td>
<td>2.62e-05</td>
</tr>
<tr>
<td>27</td>
<td>8.9979049e+02</td>
<td>8.9138752e+02</td>
<td>5.14e-07</td>
<td>1.88e-11</td>
<td>2.54e-05</td>
</tr>
<tr>
<td>28</td>
<td>8.9979041e+02</td>
<td>8.9139850e+02</td>
<td>5.13e-07</td>
<td>2.18e-11</td>
<td>2.54e-05</td>
</tr>
<tr>
<td>29</td>
<td>9.0067378e+02</td>
<td>8.9385969e+02</td>
<td>2.46e-07</td>
<td>1.46e-11</td>
<td>1.90e-05</td>
</tr>
<tr>
<td>30</td>
<td>9.0112149e+02</td>
<td>8.9746581e+02</td>
<td>2.12e-07</td>
<td>1.71e-11</td>
<td>9.61e-06</td>
</tr>
</tbody>
</table>
Primal crossover.
  Primal: Fixing 13 variables.
    12 PMoves: Infeasibility 1.97677059e-06 Objective 9.01161542e+02
    0 PMoves: Infeasibility 0.00000000e+00 Objective 9.01161540e+02
  Primal: Pushed 1, exchanged 12.
  Dual: Fixing 3 variables.
    2 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
    0 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
  Dual: Pushed 3, exchanged 0.
Using devex.
Total crossover time = 0.02 sec.

Optimal solution found.

Objective : 901.161540

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage. CPLEX logs an asterisk (*) in the left-most column for any node where it finds an integer-feasible solution or new incumbent. The + denotes an incumbent generated by the heuristic.

Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time = 0.00 sec.

Iteration log . . .
Iteration:  1  Dual objective  =  0.000000
Root relaxation solution time =  0.01 sec.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Cuts/</th>
<th>Best Node</th>
<th>ItCnt</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node</td>
<td>Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0000</td>
<td>24</td>
<td>0.0000</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
<td>6.0000</td>
<td>0</td>
<td>6.0000</td>
<td>0.0000</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
<td>4.0000</td>
<td>0</td>
<td>4.0000</td>
<td>0.0000</td>
<td>691</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>99</td>
<td>2.0000</td>
<td>15</td>
<td>4.0000</td>
<td>0.4000</td>
<td>1448</td>
</tr>
</tbody>
</table>

Fixing integer variables, and solving final LP. .
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time =  0.00 sec.

Solution satisfies tolerances.

MIP Solution : 4.000000 (2650 iterations, 185 nodes)
Final LP : 4.000000 (0 iterations)
5.11 Best integer solution possible: 1.000000
Absolute gap: 3
Relative gap: 1.5

5.11.8 Detailed Descriptions of CPLEX Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the options file is cplex.opt. The options file is case insensitive and the keywords should be given in full.

**advind** (integer): advanced basis use

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS BRatio option can be used to specify when not to use an advanced basis. The Cplex option AdvInd can be used to ignore a basis passed on by GAMS (it overrides BRatio).

Default: determined by GAMS Bratio

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use advanced basis</td>
</tr>
<tr>
<td>1</td>
<td>Use advanced basis if available</td>
</tr>
<tr>
<td>2</td>
<td>Crash an advanced basis if available (use basis with presolve)</td>
</tr>
</tbody>
</table>

**aggcutlim** (integer): aggregation limit for cut generation

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.

Default: 3

**aggfill** (integer): aggregator fill parameter

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the AggFill parameter, the substitution will not be made.

Default: 10

**aggind** (integer): aggregator on/off

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Once for LP, unlimited for MIP</td>
</tr>
<tr>
<td>0</td>
<td>Do not use</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Aggregator will be applied the specified number of times</td>
</tr>
</tbody>
</table>
auxrootthreads (integer): number of threads for auxiliary tasks at the root node

Partitions the number of threads for CPLEX to use for auxiliary tasks while it solves the root node of a problem. On a system that offers $N$ processors or $N$ global threads, if you set this parameter to $n$, where $N>n>0$ then CPLEX uses at most $n$ threads for auxiliary tasks and at most $N-n$ threads to solve the root node. See also the parameter Threads.

You cannot set $n$, the value of this parameter, to a value greater than or equal to $N$, the number of processors or global threads offered on your system. In other words, when you set this parameter to a value other than its default, that value must be strictly less than the number of processors or global threads on your system. Independent of the auxiliary root threads parameter, CPLEX will never use more threads than those defined by the global default thread count parameter. CPLEX also makes sure that there is at least one thread available for the main root tasks. For example, if you set the global threads parameter to 3 and the auxiliary root threads parameter to 4, CPLEX still uses only two threads for auxiliary root tasks in order to keep one thread available for the main root tasks. At its default value, 0 (zero), CPLEX automatically chooses the number of threads to use for the primary root tasks and for auxiliary tasks. The number of threads that CPLEX uses to solve the root node depends on several factors: 1) the number of processors available on your system; 2) the number of threads available to your application on your system (for example, as a result of limited resources or competition with other applications); 3) the value of the global default thread count parameter Threads.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Off: do not use additional threads for auxiliary tasks</td>
</tr>
<tr>
<td>0</td>
<td>Automatic: let CPLEX choose the number of threads to use</td>
</tr>
<tr>
<td>$N&gt;n&gt;0$</td>
<td>Use $n$ threads for auxiliary root tasks</td>
</tr>
</tbody>
</table>

baralg (integer): algorithm selection

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for LPs and MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Same as 1 for LPs and MIP subproblems, 3 otherwise</td>
</tr>
<tr>
<td>1</td>
<td>Infeasibility-estimate start</td>
</tr>
<tr>
<td>2</td>
<td>Infeasibility-constant start</td>
</tr>
<tr>
<td>3</td>
<td>standard barrier algorithm</td>
</tr>
</tbody>
</table>

barcolnz (integer): dense column handling

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns to be considered as dense.

Default: 0

barcrossalg (integer): barrier crossover method
Selects which crossover method is used at the end of a barrier optimization. To turn off crossover set `SolutionType` to 2.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal crossover</td>
</tr>
<tr>
<td>2</td>
<td>Dual crossover</td>
</tr>
</tbody>
</table>

`barrdisplay` (integer): progress display level

Determines the level of progress information to be displayed while the barrier method is running.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No progress information</td>
</tr>
<tr>
<td>1</td>
<td>Display normal information</td>
</tr>
<tr>
<td>2</td>
<td>Display diagnostic information</td>
</tr>
</tbody>
</table>

`barepcomp` (real): convergence tolerance

Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will terminate with an optimal solution if the relative complementarity is smaller than this value.

Default: 1.0e-08

`bargrowth` (real): unbounded face detection

Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem does have an unbounded face.

Default: 1.0e+12

`baritlim` (integer): iteration limit

Determines the maximum number of iterations for the barrier algorithm. When set to 0, no Barrier iterations occur, but problem setup occurs and information about the setup is displayed (such as Cholesky factorization information). When left at the default value, there is no explicit limit on the number of iterations.

Default: `large`

`barmaxcor` (integer): maximum correction limit

Specifies the maximum number of centering corrections that should be done on each iteration. Larger values may improve the numerical performance of the barrier algorithm at the expense of computation time. The default of -1 means the number is automatically determined.

Default: -1
**barobjrng** *(real): maximum objective function* →

Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.

Default: 1.0e+20

**barorder** *(integer): row ordering algorithm selection* →

Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of longer ordering run times. The automatic option includes additional processing and may yield results that differ from the explicit choice.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Approximate Minimum Degree (AMD)</td>
</tr>
<tr>
<td>2</td>
<td>Approximate Minimum Fill (AMF)</td>
</tr>
<tr>
<td>3</td>
<td>Nested Dissection (ND)</td>
</tr>
</tbody>
</table>

**barqcpepcomp** *(real): convergence tolerance for the barrier optimizer for QCPs* →

Range: [1.0e-12, 1.0e+75]

Default: 1.0e-07

**barstartalg** *(integer): barrier starting point algorithm* →

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the **PreDual** parameter.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>default primal, dual is 0</td>
</tr>
<tr>
<td>2</td>
<td>default primal, estimate dual</td>
</tr>
<tr>
<td>3</td>
<td>primal average, dual is 0</td>
</tr>
<tr>
<td>4</td>
<td>primal average, estimate dual</td>
</tr>
</tbody>
</table>

**bbinterval** *(integer): best bound interval* →

Set interval for selecting a best bound node when doing a best estimate search. Active only when **NodeSel** is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting **NodeSel** to 1.

Default: 7
**bendersfeascuttol** (real): Tolerance for whether a feasibility cut has been violated in Benders decomposition

Default: $1.0 \times 10^{-6}$

**bendersoptcuttol** (real): Tolerance for optimality cuts in Benders decomposition

Default: $1.0 \times 10^{-6}$

**.benderspartition** (integer): Benders partition

Default: 0

**benderspartitioninstage** (boolean): Benders partition through stage variable suffix

Default: 0

**bendersstrategy** (integer): Benders decomposition algorithm as a strategy

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly multiple) subproblems. To do so, CPLEX can make use of annotations that you supply for your model. The strategy can be applied to mixed-integer linear programs (MILP). For certain types of problems, this approach offers significant performance improvements as subproblems can be solved in parallel.

For mixed integer programs (MIP), under certain conditions, CPLEX can apply Benders algorithm to improve the search to find more feasible solutions more quickly.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Off</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Apply user annotations</td>
</tr>
<tr>
<td>2</td>
<td>Apply user annotations with automatic support for subproblems</td>
</tr>
</tbody>
</table>
value | meaning
---|---
3 | Apply automatic decomposition. CPLEX ignores any annotation supplied with the model; CPLEX applies presolve; CPLEX then automatically generates a Benders partition, putting integer variables in master and continuous linear variables into disjoint subproblems. CPLEX then solves the Benders decomposition of the model. If the problem is a strictly linear program (LP), that is, there are no integer-constrained variables to put into master, then CPLEX reports an error. If the problem is a mixed integer linear program (MILP) where all variables are integer-constrained, (that is, there are no continuous linear variables to decompose into disjoint subproblems) then CPLEX reports an error.

bndrng (string): do lower / upper bound ranging

Calculate sensitivity ranges for the specified GAMS lower and upper bounds. Unlike most options, BNDRng can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS lower and upper bound named. Specifying all will cause range information to be produced for all lower and upper bounds. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

bndstrenind (integer): bound strengthening

Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determine automatically</td>
</tr>
<tr>
<td>0</td>
<td>Don’t use bound strengthening</td>
</tr>
<tr>
<td>1</td>
<td>Use bound strengthening</td>
</tr>
</tbody>
</table>

bqpcuts (integer): boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate BQP cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate BQP cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate BQP cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate BQP cuts very aggressively</td>
</tr>
</tbody>
</table>

brdir (integer): set branching direction

Used to decide which branch (up or down) should be taken first at each node.

Default: 0
### bttol (real): backtracking limit

This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the `bttol` parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used only once a first integer solution is found or when a cutoff has been specified.

Range: $[0.0, 1.0]$

Default: 1.0

### calcqcpduals (integer): calculate the dual values of a quadratically constrained problem

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not calculate dual values</td>
</tr>
<tr>
<td>1</td>
<td>Calculate dual values as long as it does not interfere with presolve reductions</td>
</tr>
<tr>
<td>2</td>
<td>Calculate dual values and disable any presolve reductions that would interfere</td>
</tr>
</tbody>
</table>

### cliques (integer): clique cut generation

Determines whether or not clique cuts should be generated during optimization.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate clique cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate clique cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate clique cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate clique cuts very aggressively</td>
</tr>
</tbody>
</table>

### clocktype (integer): clock type for computation time

Decides how computation times are measured for both reporting performance and terminating optimization when a time limit has been set. Small variations in measured time on identical runs may be expected on any computer system with any setting of this parameter.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>CPU time</td>
</tr>
<tr>
<td>2</td>
<td>Wall clock time</td>
</tr>
</tbody>
</table>
**clonelog (integer):** enable clone logs

The clone logs contain information normally recorded in the ordinary log file but inconvenient to send through the normal log channel in case of parallel execution. The information likely to be of most interest to you are special messages, such as error messages, that result from calls to the LP optimizers called for the subproblems. The clone log files are named cloneK.log, where $K$ is the index of the clone, ranging from 0 (zero) to the number of threads minus one. Since the clones are created at each call to a parallel optimizer and discarded when it exits, the clone logs are opened at each call and closed at each exit. The clone log files are not removed when the clones themselves are discarded.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Clone log files off</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Clone log files on</td>
</tr>
</tbody>
</table>

**coeredind (integer):** coefficient reduction on/off

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of non-integral vertexes. However, the linear programs generated at each node may become more difficult to solve.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Do not use coefficient reduction</td>
</tr>
<tr>
<td>1</td>
<td>Reduce only to integral coefficients</td>
</tr>
<tr>
<td>2</td>
<td>Reduce all potential coefficients</td>
</tr>
<tr>
<td>3</td>
<td>Reduce aggressively with tilting</td>
</tr>
</tbody>
</table>

**confictalg (integer):** algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model

Default: 0

**conflictdisplay (integer):** decides how much information CPLEX reports when the conflict refiner is working

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output</td>
</tr>
<tr>
<td>1</td>
<td>Summary display</td>
</tr>
<tr>
<td>2</td>
<td>Detailed display</td>
</tr>
</tbody>
</table>

**covers (integer):** cover cut generation

Determines whether or not cover cuts should be generated during optimization.
Default: 0
The value of this parameter serves as a switch to turn on (or to turn off) CPLEX binding of multiple threads to multiple processors on platforms where this feature is available. Hexadecimal values of this parameter serve as a mask to specify to CPLEX which processors (or cores) to use in binding multiple threads to multiple processors. CPU binding is also sometimes known as processor affinity. CPU binding reduces the variability of CPLEX runs. On some occasions, running the same CPLEX on the same (non trivial) models would produce a big variation in runtime, e.g. 1000 seconds versus 900 seconds on a 12 core machine. These differences happen while CPLEX still gets exactly the same results and executes the exact same path, thanks to its completely deterministic algorithms. Running the same tests with CPU binding enabled reduced this variability in running time significantly.

If not set to off or auto CPLEX treats the value of this parameter as a string that resembles a hexadecimal number without the usual 0x prefix. A valid string consists of these elements: a) any digit from 0 (zero) through 9 (inclusive), b) any lower case character in the range a through f (inclusive), and c) any upper case character in the range A through F (inclusive). CPLEX rejects a string containing any other digits or characters than those.

When the value of this parameter is a valid string, each bit of this string corresponds to a central processing unit (CPU), that is, to a processor or core. The lowest order bit of the string corresponds to the first logical CPU, and the highest order corresponds to the last logical CPU. For example, 00000001 designates processor #0, 00000003 designates processors #0 and #1, FFFFFFFF designates all processors #0 through #31. CPLEX uses the ith CPU if and only if the ith bit of this string is set to 1 (one). Tip: For GNU/Linux users, this parameter behaves like the taskset command (except that this parameter lacks the prefix 0x).

If this CPU mask parameter is set to a valid string that designates a hexadecimal number, but global Threads count is set to 0 (zero), then CPLEX still starts as many threads as the number of cores on the machine, but only the cores enabled in the mask will be used.

For example, if a user sets this CPU mask parameter to the hexadecimal value "f" on a 16-core machine, and the user sets the global Threads count to 0 (zero), the result is 16 threads. These 16 threads will be bound to the first four cores in a round-robin way: threads 1,5,9,13 to core 1, threads 2,6,10,14 to core 2 and so on. This situation is probably not what the user intended. Therefore, if you set this CPU mask parameter, then you should also set global threads count; indeed, you should set the threads parameter to the number of active cores designated by the mask.

For example, on a 16 core machine, consider the difference between the value "off" and the value ffff. If the value of this parameter is "off" CPLEX does no binding. If the value of this parameter is ffff, CPLEX binds threads to cores.

Default: auto
**craind (integer):** crash strategy (used to obtain starting basis) ➞

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Primal: alternate ways of using objective coefficients. Dual: aggressive starting basis</td>
</tr>
<tr>
<td>0</td>
<td>Primal: ignore objective coefficients during crash. Dual: aggressive starting basis</td>
</tr>
<tr>
<td>1</td>
<td>Primal: alternate ways of using objective coefficients. Dual: default starting basis</td>
</tr>
</tbody>
</table>

**cutlo (real):** lower cutoff for tree search ➞

Sets the lower cutoff tolerance. When the problem is a maximization problem, CPLEX cuts off or discards solutions that are less than the specified cutoff value. If the model has no solution with an objective value greater than or equal to the cutoff value, then CPLEX declares the model infeasible. In other words, setting the lower cutoff value \( c \) for a maximization problem is similar to adding this constraint to the objective function of the model: \( \text{obj} \geq c \).

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: \(-1.0e+75\)

**cutpass (integer):** maximum number of cutting plane passes ➞

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Maximum passes to perform</td>
</tr>
</tbody>
</table>

**cuts (string):** default cut generation ➞

Allows generation setting of all optional cuts at once. This is done by changing the meaning of the default value (0: automatic) for the various Cplex cut generation options. The options affected are Cliques, Covers, DisjCuts, FlowCovers, FlowPaths, FracCuts, GUBCovers, ImplBd, LiftProjCuts, MCFCuts, MIDCuts, and Symmetry.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
</tbody>
</table>
### cutsfactor (real): cut limit

This option limits the number of cuts that can be added. For values between zero and one inclusive (that is, in the range \([0.0, 1.0]\), CPLEX generates no cuts.

For values strictly greater than 1.0 (one), CPLEX limits the number of rows in the model with cuts added.

The limit on this total is the product of \textit{CutsFactor} times the original number of rows. If CPLEX has presolved the model, the \textit{original number of rows} is the number of rows in the presolved model. (This behavior with respect to a presolved model is unchanged.)

CPLEX regards negative values of this parameter as equivalent to the default value -1.0. That is, a negative value specifies no particular limit on the number of cuts. CPLEX computes and dynamically adjusts such a limit automatically.

Default: \(-1.0\)

### cutup (real): upper cutoff for tree search

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off or discards any solutions that are greater than the specified upper cutoff value. If the model has no solution with an objective value less than or equal to the cutoff value, CPLEX declares the model infeasible. In other words, setting an upper cutoff value \(c\) for a minimization problem is similar to adding this constraint to the objective function of the model: \(\text{obj} \leq c\).

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: \(1.0e+75\)

### datacheck (integer): controls data consistency checking and modeling assistance

When the value of this parameter is set to level 2, CPLEX turns on both data consistency checking and modeling assistance. At this level, CPLEX issues warnings at the start of the optimization about disproportionate values (too large, too small) in coefficients, bounds, and right-hand sides (RHS).

Default: 0
**depind** (*integer*): dependency checker on/off

This option determines if and when the dependency checker will be used.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Turn off dependency checking</td>
</tr>
<tr>
<td>1</td>
<td>Turn on only at the beginning of preprocessing</td>
</tr>
<tr>
<td>2</td>
<td>Turn on only at the end of preprocessing</td>
</tr>
<tr>
<td>3</td>
<td>Turn on at the beginning and at the end of preprocessing</td>
</tr>
</tbody>
</table>

**dettilim** (*real*): deterministic time limit

Sets a time limit expressed in ticks, a unit to measure work done deterministically.

The length of a deterministic tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high.

Default: 1.0e+75

**disjcuts** (*integer*): disjunctive cuts generation

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate disjunctive cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate disjunctive cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate disjunctive cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate disjunctive cuts very aggressively</td>
</tr>
</tbody>
</table>

**divetype** (*integer*): MIP dive strategy

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting chooses when to perform a probing dive, and the other two settings direct Cplex when to perform probing dives: never or always.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Traditional dive</td>
</tr>
<tr>
<td>2</td>
<td>Probing dive</td>
</tr>
<tr>
<td>3</td>
<td>Guided dive</td>
</tr>
</tbody>
</table>
.divflt (real): solution pool range filter coefficients

A diversity filter for a solution pool (see option SolnPoo1) allows you generate solutions that
are similar to (or different from) a set of reference values that you specify for a set of binary
variables. In particular, you can use a diversity filter to generate more solutions that are
similar to an existing solution or to an existing partial solution.

A diversity filter drives the search for multiple solutions toward new solutions that satisfy
a measure of diversity specified in the filter. This diversity measure applies only to binary
variables. Potential new solutions are compared to a reference set. This reference set is
specified with this dot option. If no reference set is specified, the difference measure will be
computed relative to the other solutions in the pool. The diversity measure is computed by
summing the pair-wise absolute differences from solution and the reference values.

Default: 0

divfltlo (real): lower bound on diversity

Please check option DivFlt for general information on a diversity filter.

If you specify a lower bound on the diversity using DivFltLo, Cplex will look for solutions
that are different from the reference values. In other words, you can say, Give me solutions
that differ by at least this amount in this set of variables.

Default: mindouble

divfltup (real): upper bound on diversity

Please check option DivFlt for general information on a diversity filter.

If you specify an upper bound on diversity DivFltUp, Cplex will look for solutions similar to
the reference values. In other words, you can say, Give me solutions that are close to this one,
within this set of variables.

Default: maxdouble

dpriind (integer): dual simplex pricing

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual
steepest-edge is particularly efficient and does not carry as much computational burden as the
primal steepest-edge pricing.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Standard dual pricing</td>
</tr>
<tr>
<td>2</td>
<td>Steepest-edge pricing</td>
</tr>
<tr>
<td>3</td>
<td>Steepest-edge pricing in slack space</td>
</tr>
<tr>
<td>4</td>
<td>Steepest-edge pricing, unit initial norms</td>
</tr>
<tr>
<td>5</td>
<td>Devex pricing</td>
</tr>
</tbody>
</table>

dynamicrows (integer): switch for dynamic management of rows

This parameter specifies how CPLEX should manage rows in the current model during dual
simplex optimization. More specifically, this parameter controls the use of the kernel simplex
method (KSM) for the dual simplex algorithm. That is, CPLEX dynamically adjusts the
dimensions of the basis matrix during execution of the dual simplex algorithm, according to
the settings of this parameter.

When the value of this parameter is -1, its default value, this parameter specifies that the
user wants CPLEX to manage rows dynamically, adjusting the dimensions of the basis matrix
during dual simplex optimization. When it is set to 0, this parameter specifies that CPLEX
must keep all rows. When it is set to 1, this parameter specifies that CPLEX can keep or
discard rows according to its internal calculations.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Keep all rows</td>
</tr>
<tr>
<td>1</td>
<td>Manage rows</td>
</tr>
</tbody>
</table>

eachcutlim (integer): sets a limit for each type of cut

This parameter allows you to set a uniform limit on the number of cuts of each type that
Cplex generates. By default, the limit is a large integer; that is, there is no effective limit by
default.

Tighter limits on the number of cuts of each type may benefit certain models. For example, a
limit on each type of cut will prevent any one type of cut from being created in such large
number that the limit on the total number of all types of cuts is reached before other types of
cuts have an opportunity to be created. A setting of 0 means no cuts.

This parameter does not influence the number of Gomory cuts. For means to control the
number of Gomory cuts, see also the fractional cut parameters: FracCand, FracCuts, and
FracPass.

Default: 2100000000

epagap (real): absolute stopping tolerance

Synonym: optca

Absolute tolerance on the gap between the best integer objective and the objective of the best
node remaining. When the value falls below the value of the epagap setting, the optimization
is stopped. This option overrides GAMS OptCA which provides its initial value.

Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can
thus influence the solution path.

Default: GAMS OptCA

epgap (real): relative stopping tolerance

Synonym: optcr

Relative tolerance on the gap between the best integer objective and the objective of the best
node remaining. When the value falls below the value of the epgap setting, the mixed integer
optimization is stopped. Note the difference in the Cplex definition of the relative tolerance
with the GAMS definition. This option overrides GAMS OptCR which provides its initial
value.

Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can
thus influence the solution path.

Range: [0.0, 1.0]

Default: GAMS OptCR
epint (real): integrality tolerance

Integality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.
Range: [0.0, 0.5]
Default: 1.0e-05

eplin (real): degree of tolerance used in linearization

Default: 0.001

epmrk (real): Markowitz pivot tolerance

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.
Range: [1.0e-04, 1.0]
Default: 0.01

epopt (real): optimality tolerance

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.
Range: [1.0e-09, 0.1]
Default: 1.0e-06

epper (real): perturbation constant

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the epper option to a level where only one perturbation is required. Any value greater than or equal to 1.0e-8 is valid.
Default: 1.0e-06

eprhs (real): feasibility tolerance

Feasibility tolerance. This specifies the degree to which a problem’s basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.
Range: [1.0e-09, 0.1]
Default: 1.0e-06

exactkappa (boolean): report exact condition number in quality report

Default: 0

feasopt (boolean): computes a minimum-cost relaxation to make an infeasible model feasible

With Feasopt turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.
Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option FeasOptMode.
Feasible relaxations are available for all problem types with the exception of quadratically constrained problems.
Default: 0
feasoptmode (integer): mode of FeasOpt

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimize sum of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of all required relaxations in first phase only</td>
</tr>
<tr>
<td>1</td>
<td>Minimize sum of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
<tr>
<td>2</td>
<td>Minimize number of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the number of constraints and bounds requiring relaxation in first phase only</td>
</tr>
<tr>
<td>3</td>
<td>Minimize number of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
<tr>
<td>4</td>
<td>Minimize sum of squares of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of squares of required relaxations in first phase only</td>
</tr>
<tr>
<td>5</td>
<td>Minimize sum of squares of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
</tbody>
</table>

.feaspref (real): feasibility preference

You can express the costs associated with relaxing a bound or right hand side value during a FeasOpt run through the .feaspref option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: 1

fixoptfile (string): name of option file which is read just before solving the fixed problem

flowcovers (integer): flow cover cut generation

Determines whether or not flow cover cuts should be generated during optimization.

Default: 0
**flowpaths (integer):** flow path cut generation

Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate flow cover cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate flow cover cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate flow cover cuts aggressively</td>
</tr>
</tbody>
</table>

**folding (integer):** LP folding will be attempted during the preprocessing phase

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Turn off folder</td>
</tr>
<tr>
<td>1</td>
<td>Moderate level of folding</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive level of folding</td>
</tr>
<tr>
<td>3</td>
<td>Very aggressive level of folding</td>
</tr>
<tr>
<td>4</td>
<td>Highly aggressive level of folding</td>
</tr>
<tr>
<td>5</td>
<td>Extremely aggressive level of folding</td>
</tr>
</tbody>
</table>

**fpheur (integer):** feasibility pump heuristic

Controls the use of the feasibility pump heuristic for mixed integer programming (MIP) models.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Turns Feasible Pump heuristic off</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Apply the feasibility pump heuristic with an emphasis on finding a feasible solution</td>
</tr>
<tr>
<td>2</td>
<td>Apply the feasibility pump heuristic with an emphasis on finding a feasible solution with a good objective value</td>
</tr>
</tbody>
</table>

**fraccand (integer):** candidate limit for generating Gomory fractional cuts

Limits the number of candidate variables for generating Gomory fractional cuts.

Default: 200
fraccuts (integer): Gomory fractional cut generation ←

Determines whether or not Gomory fractional cuts should be generated during optimization.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate Gomory fractional cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate Gomory fractional cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate Gomory fractional cuts aggressively</td>
</tr>
</tbody>
</table>

fracpass (integer): maximum number of passes for generating Gomory fractional cuts ←

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter FracCuts is set to a nonzero value.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 Automatically determined</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Maximum passes to perform</td>
</tr>
</tbody>
</table>

freegamsmodel (boolean): preserves memory by dumping the GAMS model instance representation temporarily to disk ←

In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with SolveLink=0 and only for models without quadratic constraints.

Default: 0

gubcovers (integer): GUB cover cut generation ←

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization. The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate GUB cover cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate GUB cover cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate GUB cover cuts aggressively</td>
</tr>
</tbody>
</table>

heurfreq (integer): heuristic frequency ←

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval. A value of 100, for example, means that heuristics are invoked every hundredth node in the tree.
Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not use the node heuristic</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Call heuristic at the requested node interval</td>
</tr>
</tbody>
</table>

**heuristicEffort** *(real)*: the effort that CPLEX spends on heuristics during a MIP solve ↔

The value is used to increase (if >1) or decrease (if <1) the effort that CPLEX spends on heuristics during a MIP solve. If set to 0, no heuristic will run.

Default: 1.0

**iafile** *(string)*: secondary option file to be read in interactive mode triggered by iatiggerfile ↔

If in interactive mode and this option is set, options will be read from the file specified by this option instead of direct user input (as described in interactive). This option file read can be triggered by interrupting Cplex with a Control-C or using the option iatiggerfile. If defined, GAMS/CPLEX looks for this file (content irrelevant) all iatriggertime seconds and if found, reads the option file iafile. The iatiggerfile is removed afterwards so it does not trigger twice.

**iatiggerfile** *(string)*: file that triggers the reading of a secondary option file in interactive mode ↔

See iafile.

**iatriggertime** *(real)*: time interval in seconds the link looks for the trigger file in interactive mode ↔

See iafile.

Default: 60

**iis** *(boolean)*: run the conflict refiner also known as IIS finder if the problem is infeasible ↔

Find an set of conflicting constraints or IIS (Irreducably Inconsistent Set) and write an conflict report to the GAMS solution listing if the model is found to be infeasible.

Default: 0

**implbd** *(integer)*: implied bound cut generation ↔

Determines whether or not implied bound cuts should be generated during optimization.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate implied bound cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate implied bound cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate implied bound cuts aggressively</td>
</tr>
</tbody>
</table>

**indicoptstrict** *(boolean)*: abort in case of an error in indicator constraint in solver option file ↔

If enabled and a variable or equation specified in an indicator constraint is not present in the model, model generation will abort with an error message. Otherwise, if this option is disabled, erroneous indicator constraints are ignored and a warning is printed.
Default: 1

**interactive (boolean):** allow interactive option setting after a Control-C

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the `cplex.opt` file. Control is returned to Cplex by entering `continue`. The optimization can be aborted by entering `abort`. This option can only be used when running from the command line. Moreover, the GAMS option `InteractiveSolver` needs to be set to 1.

Default: 0

**intsollim (integer):** maximum number of integer solutions

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

Default: `large`

**itlim (integer):** iteration limit

Synonym: `iterlim`

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS `IterLim` option. Any non-negative integer value is valid.

Default: `GAMS IterLim`

**.lazy (boolean):** Lazy constraints activation

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraint, the solution is discarded and one or more of the violated lazy constraints are pulled into the active model.

Lazy constraints are only active if option `LazyConstraints` is enabled and are specified through the option `.lazy`. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: 0

**lazysolver (boolean):** Indicator to use lazy constraints

Default: 0

**lbheur (boolean):** local branching heuristic

This parameter lets you control whether Cplex applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, Cplex will invoke a local branching heuristic only when it finds a new incumbent. If Cplex finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

Default: 0
liftprojcuts \((\text{integer})\): lift-and-project cuts

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate lift-and-project cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate lift-and-project cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate lift-and-project cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate lift-and-project cuts very aggressively</td>
</tr>
</tbody>
</table>

localimplied \((\text{integer})\): generation of locally valid implied bound cuts

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate locally valid implied bound cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate locally valid implied bound cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate locally valid implied bound cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate locally valid implied bound cuts very aggressively</td>
</tr>
</tbody>
</table>

lowerobjstop \((\text{real})\): in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value

Default: \(-1e75\)

lpmethod \((\text{integer})\): algorithm to be used for LP problems

Specifies which LP algorithm to use. If left at the default value (0 for automatic), and a primal-feasible basis is available, primal simplex will be used. If no primal-feasible basis is available, and Threads is equal to 1, dual simplex will be used. If Threads is greater than 1 and no primal-feasible basis is available, the concurrent option will be used.

Sifting may be useful for problems with many more variables than equations.

The concurrent option runs multiple methods in parallel. The first thread uses dual simplex. The second thread uses barrier. The next thread uses primal simplex. Remaining threads are used by the barrier run. If the aspect ratio (number of columns versus number of rows) is large, and if more than 10 threads are available, then concurrent optimization also invokes sifting on the LP. The solution is returned by first method to finish.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal Simplex</td>
</tr>
</tbody>
</table>
**ltol (real)**: basis identification primal tolerance

Default: 0

**mcfcuts (integer)**: multi-commodity flow cut generation

Specifies whether Cplex should generate multi-commodity flow (MCF) cuts in a problem where Cplex detects the characteristics of a multi-commodity flow network with arc capacities. By default, Cplex decides whether or not to generate such cuts. To turn off generation of such cuts, set this parameter to -1. Cplex is able to recognize the structure of a network as represented in many real-world models. When it recognizes such a network structure, Cplex is able to generate cutting planes that usually help solve such problems. In this case, the cuts that Cplex generates state that the capacities installed on arcs pointing into a component of the network must be at least as large as the total flow demand of the component that cannot be satisfied by flow sources within the component.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate MCF cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate MCF cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate MCF cuts aggressively</td>
</tr>
</tbody>
</table>

**memoryemphasis (boolean)**: reduces use of memory

This parameter lets you indicate to Cplex that it should conserve memory where possible. When you set this parameter to its non default value, Cplex will choose tactics, such as data compression or disk storage, for some of the data computed by the barrier and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not conserve memory</td>
</tr>
<tr>
<td>1</td>
<td>Conserve memory where possible</td>
</tr>
</tbody>
</table>

**mipdisplay (integer)**: progress display level

The amount of information displayed during MIP solution increases with increasing values of this option.

Default: 4
### 5.11 CPLEX

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No display</td>
</tr>
<tr>
<td>1</td>
<td>Display integer feasible solutions</td>
</tr>
<tr>
<td>2</td>
<td>Displays nodes under mipinterval control</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2 but adds information on cuts</td>
</tr>
<tr>
<td>4</td>
<td>Same as 3 but adds LP display for the root node</td>
</tr>
<tr>
<td>5</td>
<td>Same as 3 but adds LP display for all nodes</td>
</tr>
</tbody>
</table>

**mipemphasis** *(integer):* MIP solution tactics

This option controls the tactics for solving a mixed integer programming problem.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Balance optimality and feasibility</td>
</tr>
<tr>
<td>1</td>
<td>Emphasize feasibility over optimality</td>
</tr>
<tr>
<td>2</td>
<td>Emphasize optimality over feasibility</td>
</tr>
<tr>
<td>3</td>
<td>Emphasize moving the best bound</td>
</tr>
<tr>
<td>4</td>
<td>Emphasize hidden feasible solutions</td>
</tr>
<tr>
<td>5</td>
<td>Find high quality feasible solutions as early as possible</td>
</tr>
</tbody>
</table>

**mipinterval** *(integer):* progress display interval

Controls the frequency of node logging when the parameter MIPDisplay is set higher than 1 (one). Frequency must be an integer; it may be 0 (zero), positive, or negative. By default, CPLEX displays new information in the node log during a MIP solve at relatively high frequency during the early stages of solving a MIP model, and adds lines to the log at progressively longer intervals as solving continues. In other words, CPLEX logs information frequently in the beginning and progressively less often as it works. When the value is a positive integer n, CPLEX displays new incumbents, plus it displays a new line in the log every n nodes. When the value is a negative integer n, CPLEX displays new incumbents, and the negative value determines how much processing CPLEX does before it displays a new line in the node log. A negative value close to zero means that CPLEX displays new lines in the log frequently. A negative value far from zero means that CPLEX displays new lines in the log less frequently. In other words, a negative value of this parameter contracts or dilates the interval at which CPLEX displays information in the node log.

Default: 0

**mipkappastats** *(integer):* MIP kappa computation

MIP kappa summarizes the distribution of the condition number of the optimal bases CPLEX encountered during the solution of a MIP model. That summary may let you know more about the numerical difficulties of your MIP model. Because MIP kappa (as a statistical distribution) requires CPLEX to compute the condition number of the optimal bases of the subproblems during branch-and-cut search, you can compute the MIP kappa only when CPLEX solves the subproblem with its simplex optimizer. In other words, in order to obtain results with this parameter, you can not use the sifting optimizer nor the barrier without crossover to solve the subproblems. See the parameters StartAlg and SubAlg.

Computing the kappa of a subproblem has a cost. In fact, computing MIP kappa for the basis matrices can be computationally expensive and thus generally slows down the solution of a
problem. Therefore, the setting 0 (automatic) tells CPLEX generally not to compute MIP kappa, but in cases where the parameter NumericalEmphasis is turned on, CPLEX computes MIP kappa for a sample of subproblems. The value 1 (sample) leads to a negligible performance degradation on average, but can slow down the branch-and-cut exploration by as much as 10% on certain models. The value 2 (full) leads to a 2% performance degradation on average, but can significantly slow the branch-and-cut exploration on certain models. In practice, the value 1 (sample) is a good trade-off between performance and accuracy of statistics. If you need very accurate statistics, then use value 2 (full).

In case CPLEX is instructed to compute a MIP kappa distribution, the parameter Quality is automatically turned on.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No MIP kappa statistics; default</td>
</tr>
<tr>
<td>0</td>
<td>Automatic: let CPLEX decide</td>
</tr>
<tr>
<td>1</td>
<td>Compute MIP kappa for a sample of subproblems</td>
</tr>
<tr>
<td>2</td>
<td>Compute MIP kappa for all subproblems</td>
</tr>
</tbody>
</table>

mipordind (boolean): priority list on/off

Synonym: prioropt

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to purchased within the factory. By assigning a higher priority to the build/no build decision variable, you can force this logic into the tree search and eliminate wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the MIPOrdInd option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.

Note: Priorities are assigned to discrete variables using the .prior suffix in the GAMS model. Lower .prior values mean higher priority. The .prioropt model suffix has to be used to signal GAMS to export the priorities to the solver.

Default: GAMS PriorOpt

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use priorities for branching</td>
</tr>
<tr>
<td>1</td>
<td>Priority orders are utilized</td>
</tr>
</tbody>
</table>

mipordtype (integer): priority order generation

This option is used to select the type of generic priority order to generate when no priority order is present.

Default: 0
mipsearch (integer): search strategy for mixed integer programs

Sets the search strategy for a mixed integer program. By default, Cplex chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>decreasing cost magnitude</td>
</tr>
<tr>
<td>2</td>
<td>increasing bound range</td>
</tr>
<tr>
<td>3</td>
<td>increasing cost per coefficient count</td>
</tr>
</tbody>
</table>

mipstart (integer): use mip starting values

This option controls the use of advanced starting values for mixed integer programs. A setting of 2 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization. For mipstart equals 1, 2, 3 or 4 fractional values are rounded to the nearest integer value if the integrality violation is larger than CPLEX's integer tolerance and smaller or equal to tryint. A partial MIP start is applied for mipstart equals 1, 3 or 4. Here, for discrete variables only integer values (after possible rounding) are added to the advanced starting values.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>do not use the values</td>
</tr>
<tr>
<td>1</td>
<td>set discrete variable values and use auto mipstart level</td>
</tr>
<tr>
<td>2</td>
<td>set all variable values and use check feasibility mipstart level</td>
</tr>
<tr>
<td>3</td>
<td>set discrete variable values and use solve fixed mipstart level</td>
</tr>
<tr>
<td>4</td>
<td>set discrete variable values and use solve sub-MIP mipstart level</td>
</tr>
<tr>
<td>5</td>
<td>set discrete variable values and use solve repair-MIP mipstart level</td>
</tr>
<tr>
<td>6</td>
<td>set discrete variable values and use no checks at all</td>
</tr>
</tbody>
</table>

mipstopexpr (string): stopping expression for branch and bound

If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: etalg, resusd, nodusd, obstof, objval. Supported operators are: +, -, *, /, ^, %, !=, ==, <, <=, >, >=, !, &&, ||, (, ), abs, ceil, exp, floor, log, log10, pow, sqrt. Example:

\[
\text{nodusd} \geq 1000 \text{ && abs(\text{object} - \text{objval}) / abs(\text{objval}) < 0.1}
\]

miptrace (string): filename of MIP trace file

More info is available in chapter Solve trace

miqcpstrat (integer): MIQCP relaxation choice
This option controls how MIQCPs are solved. For some models, the setting 2 may be more effective than 1. You may need to experiment with this parameter to determine the best setting for your model.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
</tbody>
</table>
| 1     | QCP relaxation  
Cplex will solve a QCP relaxation of the model at each node. |
| 2     | LP relaxation  
Cplex will solve a LP relaxation of the model at each node. |

mircuts (integer): mixed integer rounding cut generation

Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate MIR cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate MIR cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate MIR cuts aggressively</td>
</tr>
</tbody>
</table>

mpslongnum (boolean): MPS file format precision of numeric output

Determines the precision of numeric output in the MPS file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use limited MPS precision</td>
</tr>
<tr>
<td>1</td>
<td>Use full-precision</td>
</tr>
</tbody>
</table>

mtol (real): basis identification dual tolerance

Default: 0

multimipstart (string): use multiple mipstarts provided via gdx files

Specifies (multiple) GDX files with values for the variables. Each file is treated as one initial guess for the MIP start. These MIP starts are added in addition to the initial guess provided by the level attribute. A MIP start GDX file can be created, for example, by using the command line option savepoint.

multobj (boolean): controls the hierarchical optimization of multiple objectives
Default: 0

**multobjdisplay (integer):** level of display during multiobjective optimization ↔

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No display</td>
</tr>
<tr>
<td>1</td>
<td>Summary display after each subproblem</td>
</tr>
<tr>
<td>2</td>
<td>Summary display after each subproblem, as well as subproblem logs</td>
</tr>
</tbody>
</table>

**multobjmethod (integer):** method used for multi-objective solves ↔

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the *LPMethod* parameter. This parameter determines the algorithm used to solve for subsequent objectives.

Default: 0

**multobjoptfiles (string):** List of option files used for individual solves within multi-objective optimization ↔

The options given by the option files in *multobjoptfiles* are applied on top of the default GAMS/CPLEX options. This includes options set by the user via the standard option file.

If the list of option files in *multobjoptfiles* is less than the number of objective functions, the default GAMS/CPLEX options (incl. user options as before) are used to solve the remaining instances. Additional option files (i.e. more than objective functions) are ignored.

Applied options can be verified by setting *multobjdisplay* to 2.

**multobjtolmip (boolean):** enables hard constraints for hierarchical optimization objectives based on degradation tolerances ↔

CPLEX supports two different strategies to handle the degradation tolerances *objnabstol* and *objnreltol* depending on the problem type (continuous or discrete), see *objnabstol*. This setting enables the discrete strategy for continuous models. Note that *objnreltol* has no effect for discrete models. Enabling this option can lead to higher solution times.

Default: 1

**names (boolean):** load GAMS names into Cplex ↔

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.

Default: 1

**netdisplay (integer):** network display level ↔

This option controls the log for network iterations.

Default: 2
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No network log.</td>
</tr>
<tr>
<td>1</td>
<td>Displays true objective values</td>
</tr>
<tr>
<td>2</td>
<td>Displays penalized objective values</td>
</tr>
</tbody>
</table>

**netepopt** *(real)*: optimality tolerance for the network simplex method

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: \([1.0\text{e}-11, 0.1]\)

Default: 1.0e-06

**neteprhs** *(real)*: feasibility tolerance for the network simplex method

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.

Range: \([1.0\text{e}-11, 0.1]\)

Default: 1.0e-06

**netfind** *(integer)*: attempt network extraction

Specifies the level of network extraction to be done.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extract pure network only</td>
</tr>
<tr>
<td>2</td>
<td>Try reflection scaling</td>
</tr>
<tr>
<td>3</td>
<td>Try general scaling</td>
</tr>
</tbody>
</table>

**netitlim** *(integer)*: iteration limit for network simplex

Iteration limit for the network simplex method.

Default: large

**netppriind** *(integer)*: network simplex pricing

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Partial pricing</td>
</tr>
<tr>
<td>2</td>
<td>Multiple partial pricing</td>
</tr>
<tr>
<td>3</td>
<td>Multiple partial pricing with sorting</td>
</tr>
</tbody>
</table>
nodecuts (integer): decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree

Default: 0

nodefileind (integer): node storage file indicator

Specifies how node files are handled during MIP processing. Used when parameter WorkMem has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node files in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter WorkDir and Cplex actively manages which nodes remain in memory for processing.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No node files</td>
</tr>
<tr>
<td>1</td>
<td>Node files in memory and compressed</td>
</tr>
<tr>
<td>2</td>
<td>Node files on disk</td>
</tr>
<tr>
<td>3</td>
<td>Node files on disk and compressed</td>
</tr>
</tbody>
</table>

nodelim (integer): maximum number of nodes to solve

Synonym: nodlim

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix. When this parameter is set to 0 (this is only possible through an option file), Cplex completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved.

Default: GAMS NodLim

nodesel (integer): node selection strategy

This option is used to set the rule for selecting the next node to process when backtracking.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Depth-first search</td>
</tr>
<tr>
<td></td>
<td>This chooses the most recently created node.</td>
</tr>
<tr>
<td>1</td>
<td>Best-bound search</td>
</tr>
<tr>
<td></td>
<td>This chooses the unprocessed node with the best objective function for</td>
</tr>
<tr>
<td></td>
<td>the associated LP relaxation.</td>
</tr>
<tr>
<td>2</td>
<td>Best-estimate search</td>
</tr>
<tr>
<td></td>
<td>This chooses the node with the best estimate of the integer objective</td>
</tr>
<tr>
<td></td>
<td>value that would be obtained once all integer infeasibilities are</td>
</tr>
<tr>
<td></td>
<td>removed.</td>
</tr>
<tr>
<td>3</td>
<td>Alternate best-estimate search</td>
</tr>
</tbody>
</table>

numericaemphasis (boolean): emphasizes precision in numerically unstable or difficult problems
This parameter lets you indicate to Cplex that it should emphasize precision in numerically
difficult or unstable problems, with consequent performance trade-offs in time and memory.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Exercise extreme caution in computation</td>
</tr>
</tbody>
</table>

**objdif** *(real):* overrides GAMS Cheat parameter

Synonym: cheat

A means for automatically updating the cutoff to more restrictive values. Normally the most
recently found integer feasible solution objective value is used as the cutoff for subsequent
nodes. When this option is set to a positive value, the value will be subtracted from (added
to) the newly found integer objective value when minimizing (maximizing). This forces the
MIP optimization to ignore integer solutions that are not at least this amount better than
the one found so far. The option can be adjusted to improve problem solving efficiency by
limiting the number of nodes; however, setting this option at a value other than zero (the
default) can cause some integer solutions, including the true integer optimum, to be missed.
Negative values for this option will result in some integer solutions that are worse than or
the same as those previously generated, but will not necessarily result in the generation of all
possible integer solutions. This option overrides the GAMS Cheat parameter.

Default: 0.0

**objllim** *(real):* objective function lower limit

Setting a lower objective function limit will cause Cplex to halt the optimization process once
the minimum objective function value limit has been exceeded.

Default: -1.0e+75

**objnabstol** *(string):* allowable absolute degradation for objective

This parameter is used to set the allowable degradation for an objective when doing hierar-
chical multi-objective optimization *(MultObj).* The syntax for this parameter is `ObjNAbsTol
ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model
one at a time, in priority order. For MIPs *(or if MultObjTolMip is enabled)*, if it achieves
objective value `z` when it optimizes for this objective, then subsequent steps are allowed to
degrade this value by at most `ObjNAbsTol`. For LPs, `ObjNAbsTol` defines a threshold for
reduced costs above which nonbasic variables in the associated LP solve will be fixed at the
bound at which they reside.

**objnrealtol** *(string):* allowable relative degradation for objective

This parameter is used to set the allowable degradation for an objective when doing hierar-
chical multi-objective optimization *(MultObj).* The syntax for this parameter is `ObjNRelTol
ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model
one at a time, in priority order. For MIPs *(or if MultObjTolMip is enabled)*, if it achieves
objective value `z` when it optimizes for this objective, then subsequent steps are allowed to
degrade this value by at most `ObjNRelTol*|z|`. This option has no effect for continuous models.
objrng (string): do objective ranging

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, ObjRng can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying all will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

Default: no objective ranging is done

objulim (real): objective function upper limit

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.

Default: 1.0e+75

optimalitytarget (integer): type of optimality that Cplex targets

This parameter specifies the type of solution that CPLEX attempts to compute with respect to the optimality of that solution when CPLEX solves a continuous (QP) or mixed integer (MIQP) quadratic model. In other words, the variables of the model can be continuous or mixed integer and continuous; the objective function includes a quadratic term, and perhaps the objective function is not positive semi-definite (non PSD). This parameter does not apply to quadratically constrained mixed integer problems (MIQCP); that is, this parameter does not apply to mixed integer problems that include a quadratic term among the constraints.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic CPLEX first attempts to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX will return with an error (Q is not PSD).</td>
</tr>
<tr>
<td>1</td>
<td>Search for a globally optimal solution to a convex model CPLEX searches for a globally optimal solution to a convex model. In problems of type QP or MIQP, this setting interacts with linearization switch QToLin for QP, MIQP</td>
</tr>
<tr>
<td>2</td>
<td>Search for a solution that satisfies first-order optimality conditions no optimality guarantee CPLEX first attempt to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX searches for a solution that satisfies first-order optimality conditions but is not necessarily globally optimal.</td>
</tr>
<tr>
<td>3</td>
<td>Search for a globally optimal solution regardless of convexity If the problem type is QP, CPLEX first changes the problem type to MIQP. CPLEX then solves the problem (whether originally QP or MIQP) to global optimality. In problems of type QP or MIQP, this setting interacts with with linearization switch QToLin for QP, MIQP. With this setting information about dual values is not available for the solution.</td>
</tr>
</tbody>
</table>

parallelmode (integer): parallel optimization mode

Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.
In this context, deterministic means that multiple runs with the same model at the same parameter settings on the same platform will reproduce the same solution path and results. In contrast, opportunistic implies that even slight differences in timing among threads or in the order in which tasks are executed in different threads may produce a different solution path and consequently different timings or different solution vectors during optimization executed in parallel threads. When running with multiple threads, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.

In deterministic mode, Cplex applies as much parallelism as possible while still achieving deterministic results. That is, when you run the same model twice on the same platform with the same parameter settings, you will see the same solution and optimization run.

More opportunities to exploit parallelism are available if you do not require determinism. In other words, Cplex can find more opportunities for parallelism if you do not require an invariant, repeatable solution path and precisely the same solution vector. To use all available parallelism, you need to select the opportunistic parallel mode. In this mode, Cplex will utilize all opportunities for parallelism in order to achieve best performance.

However, in opportunistic mode, the actual optimization may differ from run to run, including the solution time itself. A truly parallel deterministic algorithm is available only for MIP optimization. Only opportunistic parallel algorithms (barrier and concurrent optimizers) are available for continuous models. (Each of the simplex algorithms runs sequentially on a continuous model.) Consequently, when parallel mode is set to deterministic, both barrier and concurrent optimizers are restricted to run only sequentially, not in parallel.

A GAMS/Cplex run will use deterministic mode unless explicitly specified.

If `ParallelMode` is explicitly set to 0 (automatic) the settings of this parallel mode parameter interact with settings of the `Threads` parameter. Let the result number of threads available to Cplex be $n$ (note that negative values for the threads parameter are possible to exclude work on some cores).

Here is is list of possible value:

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Enable opportunistic parallel search mode</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Enable deterministic parallel search mode</td>
</tr>
</tbody>
</table>

`perind (boolean): force initial perturbation ←`

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting `PerInd` to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not automatically perturbed</td>
</tr>
<tr>
<td>1</td>
<td>automatically perturbed</td>
</tr>
</tbody>
</table>
**perlim (integer):** number of stalled iterations before perturbation

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.

Default: 0

**polishafterdett ime (real):** deterministic time before starting to polish a feasible solution

Default: $1.0 \times 10^{75}$

**polishafterepgap (real):** absolute MIP gap before starting to polish a feasible solution

Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch-and-cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch-and-cut, or a MIP start supplied by a user. Because of the high cost entailed by solution polishing, it is not called throughout branch-and-cut like other heuristics. Instead, solution polishing works in a second phase after a first phase of conventional branch-and-cut. As an additional step after branch-and-cut, solution polishing can improve the best known solution. As a kind of branch-and-cut algorithm itself, solution polishing focuses solely on finding better solutions. Consequently, it may not prove optimality, even if the optimal solution has indeed been found. Like the RINS heuristic, solution polishing explores neighborhoods of previously found solutions by solving subMIPs.

Sets an absolute MIP gap (that is, the difference between the best integer objective and the objective of the best node remaining) after which CPLEX stops branch-and-cut and begins polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 0.0

**polishafterepgap (real):** relative MIP gap before starting to polish a solution

Sets a relative MIP gap after which CPLEX will stop branch-and-cut and begin polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 0.0

**polishafterintsol (integer):** MIP integer solutions to find before starting to polish a feasible solution

Sets the number of integer solutions to find before CPLEX stops branch-and-cut and begins to polish a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 2147483647

**polishafternode (integer):** nodes to process before starting to polish a feasible solution

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible solution is available.

Default: 2147483647

**polishaftertime (real):** time before starting to polish a feasible solution
Tells CPLEX how much time in seconds to spend during mixed integer optimization before CPLEX starts polishing a feasible solution. The default value is such that CPLEX does not start solution polishing by default.

Default: 1.0e+75

**populatelim (integer):** limit of solutions generated for the solution pool by populate method

Limits the number of solutions generated for the solution pool during each call to the populate procedure. Populate stops when it has generated `PopulateLim` solutions. A solution is counted if it is valid for all filters (see `DivFlt` and consistent with the relative and absolute pool gap parameters (see `SlnPoolGap` and `SlnPoolAGap`), and has not been rejected by the incumbent checking routine (see `UserIncbCall`), whether or not it improves the objective of the model. This parameter does not apply to MIP optimization generally; it applies only to the populate procedure.

If you are looking for a parameter to control the number of solutions stored in the solution pool, consider the parameter `SlnPoolCapacity` instead.

Populate will stop before it reaches the limit set by this parameter if it reaches another limit, such as a time or node limit set by the user.

Default: 20

**pPriInd (integer):** primal simplex pricing

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use `DPriInd` to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
</table>
| -1    | Reduced-cost pricing  
This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column). |
| 0     | Hybrid reduced-cost and Devex pricing |
| 1     | Devex pricing  
This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous. |
| 2     | Steepest edge pricing  
If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems. |
| 3     | Steepest edge pricing with slack initial norms  
This reduces the computationally intensive nature of steepest edge pricing. |
| 4     | Full pricing |
**predual (integer):** give dual problem to the optimizer

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The `PreDual` option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>do not give dual to optimizer</td>
</tr>
<tr>
<td>0</td>
<td>automatic</td>
</tr>
<tr>
<td>1</td>
<td>give dual to optimizer</td>
</tr>
</tbody>
</table>

**preind (boolean):** turn presolver on/off

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, it may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.

Default: 1

**prepass (integer):** number of presolve applications to perform

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set `PreInd` to 0 to turn both off.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>0</td>
<td>No presolve</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of MIP presolve applications to perform</td>
</tr>
</tbody>
</table>

**prereform (integer):** set presolve reformulations

Default: 3

**preslvnd (integer):** node presolve selector

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No node presolve</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Force node presolve</td>
</tr>
<tr>
<td>2</td>
<td>Perform probing on integer-infeasible variables</td>
</tr>
<tr>
<td>3</td>
<td>Perform aggressive node probing</td>
</tr>
</tbody>
</table>
**pricelim** *(integer)*: pricing candidate list

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative integer values are valid.

Default: 0, in which case it is determined automatically

**printoptions** *(boolean)*: list values of all options to GAMS listing file

Write the values of all options to the GAMS listing file. Valid values are no or yes.

Default: 0

**probe** *(integer)*: perform probing before solving a MIP

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No probing</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Limited probing</td>
</tr>
<tr>
<td>2</td>
<td>More probing</td>
</tr>
<tr>
<td>3</td>
<td>Full probing</td>
</tr>
</tbody>
</table>

**probedettime** *(real)*: deterministic time spent probing

Default: 1.0e+75

**probetime** *(real)*: time spent probing

Limits the amount of time in seconds spent probing.

Default: 1.0e+75

**qextractalg** *(integer)*: quadratic extraction algorithm in GAMS interface

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.</td>
</tr>
<tr>
<td>2</td>
<td>DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.</td>
</tr>
</tbody>
</table>

**qpmakepsdind** *(boolean)*: adjust MIQP formulation to make the quadratic matrix positive-semi-definite


Determines whether Cplex will attempt to adjust a MIQP formulation, in which all the variables appearing in the quadratic term are binary. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (PSD, as required by Cplex for all QP formulations), to make it PSD, and will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 means yes but you can turn it of if necessary; most models should benefit from the default setting.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

**qpmethod (integer):** algorithm to be used for QP problems

Specifies which QP algorithm to use.

At the default of 0 (automatic), barrier is used for QP problems and dual simplex for the root relaxation of MIQP problems.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal Simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual Simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network Simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier</td>
</tr>
<tr>
<td>5</td>
<td>Sifting</td>
</tr>
<tr>
<td>6</td>
<td>Concurrent dual, barrier, and primal</td>
</tr>
</tbody>
</table>

**qtolin (integer):** linearization of the quadratic terms in the objective function of a QP or MIQP model

This parameter switches on or off linearization of the quadratic terms in the objective function of a quadratic program or of a mixed integer quadratic program.

In a convex mixed integer quadratic program, this parameter controls whether Cplex linearizes the product of binary variables in the objective function during presolve. In a nonconvex quadratic program or mixed integer quadratic program solved to global optimality according to OptimalityTarget, this parameter controls how Cplex linearizes the product of bounded variables in the objective function during presolve.

This parameter interacts with the existing parameter OptimalityTarget: When the solution target type is set to 1 (that is, Cplex searches for a globally optimal solution to a convex model), then in a convex MIQP, this parameter tells Cplex to replace the product of a binary variable and a bounded linear variable by a linearly constrained variable. When the solution target type is set to 3, then in a nonconvex QP or nonconvex MIQP, this parameter controls the initial relaxation.

Default: −1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Off, Cplex does not linearize quadratic terms in the objective</td>
</tr>
<tr>
<td>1</td>
<td>On, Cplex linearizes quadratic terms in the objective</td>
</tr>
</tbody>
</table>
**quality** (boolean): write solution quality statistics

Write solution quality statistics to the listing and log file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing and contain information about infeasibility levels, solution value magnitudes, and the condition number (kappa):

Solution Quality Statistics:

<table>
<thead>
<tr>
<th></th>
<th>unscaled</th>
<th></th>
<th>scaled</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>sum</td>
<td>max</td>
<td>sum</td>
</tr>
<tr>
<td>primal infeasibility</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>dual infeasibility</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>primal residual</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>dual residual</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
</tr>
<tr>
<td>primal solution vector</td>
<td>3.000e+02</td>
<td>9.000e+02</td>
<td>3.000e+02</td>
<td>9.000e+02</td>
</tr>
<tr>
<td>dual solution vector</td>
<td>1.000e+00</td>
<td>1.504e+00</td>
<td>1.000e+00</td>
<td>1.504e+00</td>
</tr>
<tr>
<td>slacks</td>
<td>5.000e+01</td>
<td>5.000e+01</td>
<td>5.000e+01</td>
<td>5.000e+01</td>
</tr>
<tr>
<td>reduced costs</td>
<td>3.600e-02</td>
<td>4.500e-02</td>
<td>3.600e-02</td>
<td>4.500e-02</td>
</tr>
</tbody>
</table>

Condition number of the scaled basis matrix = 9.000e+00

Default: 0

**randomseed** (integer): sets the random seed differently for diversity of solutions

Default: changes with each Cplex release

**readflt** (string): reads Cplex solution pool filter file

The GAMS/Cplex solution pool options cover the basic use of diversity and range filters for producing multiple solutions. If you need multiple filters, weights on diversity filters or other advanced uses of solution pool filters, you could produce a Cplex filter file with your favorite editor or the GAMS Put Facility and read this into GAMS/Cplex using this option.

**readparams** (string): read Cplex parameter file

**reduce** (integer): primal and dual reduction type

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No primal or dual reductions</td>
</tr>
<tr>
<td>1</td>
<td>Only primal reductions</td>
</tr>
<tr>
<td>2</td>
<td>Only dual reductions</td>
</tr>
<tr>
<td>3</td>
<td>Both primal and dual reductions</td>
</tr>
</tbody>
</table>

**reinv** (integer): refactorization frequency

Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex's
performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.

Default: 0, in which case it is determined automatically

relaxfixedinfeas (boolean): accept small infeasibilities in the solve of the fixed problem

Sometimes the solution of the fixed problem of a MIP does not solve to optimality due to small (dual) infeasibilities. The default behavior of the GAMS/Cplex link is to return the primal solution values only. If the option is set to 1, the small infeasibilities are ignored and a full solution including the dual values are reported back to GAMS.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

relaxpreind (integer): presolve for initial relaxation on/off

This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>do not presolve initial relaxation</td>
</tr>
<tr>
<td>1</td>
<td>use presolve on initial relaxation</td>
</tr>
</tbody>
</table>

relobjdif (real): relative cheat parameter

The relative version of the ObjDif option. Ignored if ObjDif is non-zero.

Default: 0.0

repairtries (integer): try to repair infeasible MIP start

This parameter lets you indicate to Cplex whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>None: do not try to repair</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Maximum tries to perform</td>
</tr>
</tbody>
</table>

repeatpresolve (integer): reapply presolve at root after preprocessing
This integer parameter tells Cplex whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Turn off represolve</td>
</tr>
<tr>
<td>1</td>
<td>Represolve without cuts</td>
</tr>
<tr>
<td>2</td>
<td>Represolve with cuts</td>
</tr>
<tr>
<td>3</td>
<td>Represolve with cuts and allow new root cuts</td>
</tr>
</tbody>
</table>

rerun (string): rerun problem if presolve infeasible or unbounded

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with presolve turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The IIS option can be used to get even more diagnostic information. The rerun option controls this behavior. Valid values are auto, yes, no and nono. The value of auto is equivalent to no if names are successfully loaded into Cplex and option IIS is set to no. In that case the Cplex messages from presolve help identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to yes. Loading of GAMS names into Cplex is controlled by option Names. The value of nono only affects MIP models for which Cplex finds a feasible solution in the branch-and-bound tree but the fixed problem turns out to be infeasible. In this case the value nono also disables the rerun without presolve, while the value of no still tries this run. Feasible integer solution but an infeasible fixed problem happens in few cases and mostly with badly scaled models. If you experience this try more aggressive scaling (ScaInd) or tightening the integer feasibility tolerance EPInt. If the fixed model is infeasible only the primal solution is returned to GAMS. You can recognize this inside GAMS by checking the marginal of the objective defining constraint which is always nonzero.

Default: yes

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Automatic</td>
</tr>
<tr>
<td>yes</td>
<td>Rerun infeasible models with presolve turned off</td>
</tr>
<tr>
<td>no</td>
<td>Do not rerun infeasible models</td>
</tr>
<tr>
<td>nono</td>
<td>Do not rerun infeasible fixed MIP models</td>
</tr>
</tbody>
</table>

rhsrng (string): do right-hand-side ranging

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, RHSRng can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS equation named. Specifying all will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

Default: no right-hand-side ranging is done

rinsheur (integer): relaxation induced neighborhood search frequency
Cplex implements a heuristic known a Relaxation Induced Neighborhood Search (RINS) for MIP and MIQCP problems. RINS explores a neighborhood of the current incumbent to try to find a new, improved incumbent. It formulates the neighborhood exploration as a MIP, a subproblem known as the subMIP, and truncates the subMIP solution by limiting the number of nodes explored in the search tree.

Parameter `RINSHeur` controls how often RINS is invoked. A value of 100, for example, means that RINS is invoked every hundredth node in the tree.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Disable RINS</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Call RINS at the requested node interval</td>
</tr>
</tbody>
</table>

`rltcuts (integer)`: Reformulation Linearization Technique (RLT) cuts

This parameter controls the addition of cuts based on the Reformulation Linearization Technique (RLT) for nonconvex quadratic programs (QP) or mixed integer quadratic programs (MIQP) solved to global optimality. That is, the `OptimalityTarget` parameter must be set to 3. The `RLTCuts` option is not controlled by the option `Cuts`.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not generate RLT cuts</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>1</td>
<td>Generate RLT cuts moderately</td>
</tr>
<tr>
<td>2</td>
<td>Generate RLT cuts aggressively</td>
</tr>
<tr>
<td>3</td>
<td>Generate RLT cuts very aggressively</td>
</tr>
</tbody>
</table>

`rngrestart (string)`: write GAMS readable ranging information file

Write ranging information, in GAMS readable format, to the file named. If the file extension is GDX, the ranging information is exported as GDX file. Options `ObjRng` and `RHSRng` are used to specify which GAMS variables or equations are included.

Default: ranging information is printed to the listing file

`scaind (integer)`: matrix scaling on/off

This option influences the scaling of the problem matrix.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No scaling</td>
</tr>
<tr>
<td>0</td>
<td>Standard scaling</td>
</tr>
<tr>
<td></td>
<td>An equilibration scaling method is implemented which is generally very effective.</td>
</tr>
<tr>
<td>1</td>
<td>Modified, more aggressive scaling method</td>
</tr>
<tr>
<td></td>
<td>This method can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.</td>
</tr>
</tbody>
</table>
siftalg *(integer)*: sifting subproblem algorithm

Sets the algorithm to be used for solving sifting subproblems.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier</td>
</tr>
</tbody>
</table>

siftdisplay *(integer)*: sifting display level

Determines the amount of sifting progress information to be displayed.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No display</td>
</tr>
<tr>
<td>1</td>
<td>Display major iterations</td>
</tr>
<tr>
<td>2</td>
<td>Display LP subproblem information</td>
</tr>
</tbody>
</table>

sifting *(boolean)*: switch for sifting from simplex optimization

Default: 1

siftitlim *(integer)*: limit on sifting iterations

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

Default: large

simdisplay *(integer)*: simplex display level

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No iteration messages are issued until the optimal solution is reported</td>
</tr>
<tr>
<td>1</td>
<td>An iteration log message will be issued after each refactorization</td>
</tr>
<tr>
<td></td>
<td>Each entry will contain the iteration count and scaled infeasibility or</td>
</tr>
<tr>
<td></td>
<td>objective value.</td>
</tr>
<tr>
<td>2</td>
<td>An iteration log message will be issued after each iteration</td>
</tr>
<tr>
<td></td>
<td>The variables, slacks and artificials entering and leaving the basis</td>
</tr>
<tr>
<td></td>
<td>will also be reported.</td>
</tr>
</tbody>
</table>

singlim *(integer)*: limit on singularity repairs
The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that has been found. Any non-negative integer value is valid.

Default: 10

**solnpool** *(string)*: solution pool file name

The solution pool enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file name contains the information about the different solutions generated by Cplex. Inside your GAMS program you can process the GDX file and read the different solution point files. Please check the GAMS/Cplex solver guide document and the example model solnpool.gms from the GAMS model library.

**solnpoolagap** *(real)*: absolute tolerance for the solutions in the solution pool

Sets an absolute tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the objective of the incumbent solution according to this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap $\text{SolnPoolGap}$ may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and also within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool absolute gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: $1.0e+75$

**solnpoolcapacity** *(integer)*: limits of solutions kept in the solution pool

Limits the number of solutions kept in the solution pool. At most, $\text{SolnPoolCapacity}$ solutions will be stored in the pool. Superfluous solutions are managed according to the replacement strategy set by the solution pool replacement parameter $\text{SolnPoolReplace}$.

The optimization (whether by MIP optimization or the populate procedure) will not stop if more than $\text{SolnPoolCapacity}$ are generated. Instead, stopping criteria are regular node and time limits and $\text{PopulateLim}$, $\text{SolnPoolGap}$ and $\text{SolnPoolAGap}$.

Default: 2100000000

**solnpoolgap** *(real)*: relative tolerance for the solutions in the solution pool

Sets a relative tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool relative gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: $1.0e+75$
**solnpoolintensity (integer):** solution pool intensity for ability to produce multiple solutions

Controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed. This parameter applies both to MIP optimization and to the populate procedure.

Values from 1 to 4 invoke increasing effort to find larger numbers of solutions. Higher values are more expensive in terms of time and memory but are likely to yield more solutions.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic&lt;br&gt;Its default value, 0, lets Cplex choose which intensity to apply.</td>
</tr>
<tr>
<td>1</td>
<td>Mild: generate few solutions quickly&lt;br&gt;For value 1, the performance of MIP optimization is not affected. There is no slowdown and no additional consumption of memory due to this setting. However, populate will quickly generate only a small number of solutions. Generating more than a few solutions with this setting will be slow. When you are looking for a larger number of solutions, use a higher value of this parameter.</td>
</tr>
<tr>
<td>2</td>
<td>Moderate: generate a larger number of solutions&lt;br&gt;For value 2, some information is stored in the branch and cut tree so that it is easier to generate a larger number of solutions. This storage has an impact on memory used but does not lead to a slowdown in the performance of MIP optimization. With this value, calling populate is likely to yield a number of solutions large enough for most purposes. This value is a good choice for most models.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive: generate many solutions and expect performance penalty&lt;br&gt;For value 3, the algorithm is more aggressive in computing and storing information in order to generate a large number of solutions. Compared to values 1 and 2, this value will generate a larger number of solutions, but it will slow MIP optimization and increase memory consumption. Use this value only if setting this parameter to 2 does not generate enough solutions.</td>
</tr>
<tr>
<td>4</td>
<td>Very aggressive: enumerate all practical solutions&lt;br&gt;For value 4, the algorithm generates all solutions to your model. Even for small models, the number of possible solutions is likely to be huge; thus enumerating all of them will take time and consume a large quantity of memory.</td>
</tr>
</tbody>
</table>

**solnpoolmerge (string):** solution pool file name for merged solutions

Similar to solnpool this option enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file contains all variables with an additional first index (determined through SolnPoolPrefix) as parameters (Cplex only reports the primal solution). Inside your GAMS program you can process the GDX file and read all solutions in one read operation. Please check the GAMS/Cplex solver guide document for further solution pool options and the example model solmpool.gms from the GAMS model library.

**solnpoolnumsym (integer):** maximum number of variable symbols when writing merged solutions

Default: 10

**solnpoolpop (integer):** methods to populate the solution pool
Regular MIP optimization automatically adds incumbents to the solution pool as they are discovered. Cplex also provides a procedure known as populate specifically to generate multiple solutions. You can invoke this procedure either as an alternative to the usual MIP optimizer or as a successor to the MIP optimizer. You can also invoke this procedure many times in a row in order to explore the solution space differently (see option SolnPoolPopRepeat). In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Just collect the incumbents found during regular optimization</td>
</tr>
<tr>
<td>2</td>
<td>Calls the populate procedure</td>
</tr>
</tbody>
</table>

solnpoolpopdel (string): file with solution numbers to delete from the solution pool

After the GAMS program specified in SolnPoolPopRepeat determined to continue the search for alternative solutions, the file specified by this option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

solnpoolpoprepeat (string): method to decide if populating the solution should be repeated

After the termination of the populate procedure (see option SolnPoolPop). The GAMS program specified in this option will be called which can examine the solutions in the solution pool and can decide to run the populate procedure again. If the GAMS program terminates normally (not compilation or execution time error) the search for new alternative solutions will be repeated.

solnpoolprefix (string): file name prefix for GDX solution files

Default: soln

solnpoolexplicit (integer): strategy for replacing a solution in the solution pool

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Replace the first solution (oldest) by the most recent solution; first in, first out</td>
</tr>
<tr>
<td>1</td>
<td>Replace the solution which has the worst objective</td>
</tr>
<tr>
<td>2</td>
<td>Replace solutions in order to build a set of diverse solutions</td>
</tr>
</tbody>
</table>

solutiontype (integer): type of solution (basic or non basic) for an LP or QP

Specifies the type of solution (basic or non basic) that CPLEX attempts to compute for a linear program (LP) or for a quadratic program (QP). In this context, basic means having to do with the basis, and non basic applies to the variables and constraints not participating in the basis.

By default (that is, when the value of this parameter is 0 (zero) automatic), CPLEX seeks a basic solution (that is, a solution with a basis) for all linear programs (LP) and for all quadratic programs (QP).

When the value of this parameter is 1 (one), CPLEX seeks a basic solution, that is, a solution that includes a basis with a basic status for variables and constraints. In other words, CPLEX behaves the same way for the values 0 (zero) and 1 (one) of this parameter.
When the value of this parameter is 2, CPLEX seeks a pair of primal-dual solution vectors. This setting does not prevent CPLEX from producing status information, but in seeking a pair of primal-dual solution vectors, CPLEX possibly may not produce basic status information; that is, it is possible that CPLEX does not produce status information about which variables and constraints participate in the basis at this setting.

Do not use the deprecated value -1 (minus one) of the parameter barrier crossover algorithm to turn off crossover of the barrier algorithm but use this parameter to indicate that a primal-dual pair is sufficient.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Basic solution</td>
</tr>
<tr>
<td>2</td>
<td>primal-dual pair</td>
</tr>
</tbody>
</table>

**solvefinal (boolean):** switch to solve the problem with fixed discrete variables

Sometimes the solution process after the branch-and-cut that solves the problem with fixed discrete variables takes a long time and the user is interested in the primal values of the solution only. In these cases, solvefinal can be used to turn this final solve off. Without the final solve no proper marginal values are available and only NAs are returned to GAMS.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not solve the fixed problem</td>
</tr>
<tr>
<td>1</td>
<td>Solve the fixed problem and return duals</td>
</tr>
</tbody>
</table>

**sos1reform (integer):** automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)

Default: 0

**sos2reform (integer):** automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)

Default: 0

**startalg (integer):** MIP starting algorithm

Selects the algorithm to use for the initial relaxation of a MIP.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier</td>
</tr>
<tr>
<td>5</td>
<td>Sifting</td>
</tr>
<tr>
<td>6</td>
<td>Concurrent</td>
</tr>
</tbody>
</table>
strongcandlim (integer): size of the candidates list for strong branching

Limit on the length of the candidate list for strong branching (VarSel = 3).

Default: 10

strongitlim (integer): limit on iterations per branch for strong branching

Limit on the number of iterations per branch in strong branching (VarSel = 3). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.

Default: 0

subalg (integer): algorithm for subproblems

Strategy for solving linear sub-problems at each node.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network optimizer followed by dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier with crossover</td>
</tr>
<tr>
<td>5</td>
<td>Sifting</td>
</tr>
</tbody>
</table>

submipnodelim (integer): limit on number of nodes in an RINS subMIP

Controls the number of nodes explored in an RINS subMIP. See option RINSHeur.

Default: 500

submipscale (integer): scale the problem matrix when CPLEX solves a subMIP during MIP optimization

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No scaling</td>
</tr>
<tr>
<td>0</td>
<td>Standard scaling</td>
</tr>
<tr>
<td>1</td>
<td>Modified, more aggressive scaling method</td>
</tr>
</tbody>
</table>

submipstartalg (integer): starting algorithm for a subMIP of a MIP

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
</tbody>
</table>
**submipsubalg** (integer): algorithm for subproblems of a subMIP of a MIP

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network optimizer followed by dual simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier with crossover</td>
</tr>
<tr>
<td>5</td>
<td>Sifting</td>
</tr>
</tbody>
</table>

**symmetry** (integer): symmetry breaking cuts

Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Turn off symmetry breaking</td>
</tr>
<tr>
<td>1</td>
<td>Moderate level of symmetry breaking</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive level of symmetry breaking</td>
</tr>
<tr>
<td>3</td>
<td>Very aggressive level of symmetry breaking</td>
</tr>
<tr>
<td>4</td>
<td>Highly aggressive level of symmetry breaking</td>
</tr>
<tr>
<td>5</td>
<td>Extremely aggressive level of symmetry breaking</td>
</tr>
</tbody>
</table>

**threads** (integer): global default thread count

Synonym: gthreads

Default number of parallel threads allowed for any solution method. Negative values are interpreted as the number of cores to leave free so setting threads to -1 leaves one core free for other tasks. Cplex does not understand negative values for the threads parameter. GAMS/Cplex will translate this into a positive number by applying the following formula: max(1, number of cores−|threads|). Setting threads to 0 lets Cplex use at most 32 threads or the number of cores of the machine, whichever is smaller.

Default: GAMS Threads

**tilim** (real): overrides the GAMS ResLim option

Synonym: reslim

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.

Default: GAMS ResLim
**trelim (real):** maximum space in memory for tree

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.

Default: $1.0e+75$

**tuning (string):** invokes parameter tuning tool

Invokes the Cplex parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Cplex option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in LP, MPS or SAV format. Cplex will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Cplex option file name without considering the problem provided by GAMS. Due to technical reasons a single option input line is limited by 256 characters. If the list of model files exceeds this length you can provide a second, third, ... line starting again with keyword tuning and a list of model instance files.

The result of such a tuning run is the updated GAMS/Cplex option file with a tuned set of parameters. The solver and model status returned to GAMS will be **NORMAL COMPLETION** and **NO SOLUTION**. More details on Cplex tuning can be found on IBM’s web page. Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

**tuningdettlim (real):** tuning deterministic time limit per model or suite

Default: $1.0e+75$

**tuningdisplay (integer):** level of information reported by the tuning tool

Specifies the level of information reported by the tuning tool as it works.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off display</td>
</tr>
<tr>
<td>1</td>
<td>Display standard minimal reporting</td>
</tr>
<tr>
<td>2</td>
<td>Display standard report plus parameter settings being tried</td>
</tr>
<tr>
<td>3</td>
<td>Display exhaustive report and log</td>
</tr>
</tbody>
</table>

**tuningmeasure (integer):** measure for evaluating progress for a suite of models

Controls the measure for evaluating progress when a suite of models is being tuned. Choices are mean average and minmax of time to compare different parameter sets over a suite of models.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mean average</td>
</tr>
<tr>
<td>2</td>
<td>minmax</td>
</tr>
</tbody>
</table>

**tuningrepeat (integer):** number of times tuning is to be repeated on perturbed versions
Specifies the number of times tuning is to be repeated on perturbed versions of a given problem. The problem is perturbed automatically by Cplex permuting its rows and columns. This repetition is helpful when only one problem is being tuned, as repeated perturbation and re-tuning may lead to more robust tuning results. This parameter applies to only one problem in a tuning session.

Default: 1

**tuningtilim** *(real)*: tuning time limit per model or suite

Sets a time limit per model and per test set (that is, suite of models).

As an example, suppose that you want to spend an overall amount of time tuning the parameter settings for a given model, say, 2000 seconds. Also suppose that you want Cplex to make multiple attempts within that overall time limit to tune the parameter settings for your model. Suppose further that you want to set a time limit on each of those attempts, say, 200 seconds per attempt. In this case you need to specify an overall time limit of 2000 using GAMS option `reslim` or Cplex option `TiLim` and `tuningtilim` to 200.

Default: 0.2×GAMS `ResLim`

**upperobjstop** *(real)*: in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value

Default: 1e75

**usercallparmfile** *(string)*: Command-line parameter include file used in GAMS command-line calls triggered by BCH

**.usercut** *(boolean)*: User cut activation

Determines whether a linear constraint is treated as a user cut. At the beginning of the MIP solution process, any constraint whose usercut attribute is set to 1 (the default value is 0) is removed from the model and placed in the user cut pool. User cuts may be used by CPLEX at any time to improve the solution process. There is no guarantee that they are actually used.

The user cut pool is only active if option `usercutpool` is enabled and are specified through the option `.usercut`. The syntax for `dotoptions` is explained in the Introduction chapter of the Solver Manual.

Default: 0

**usercutcall** *(string)*: the GAMS command line to call the cut generator

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

**usercutfirst** *(integer)*: calls the cut generator for the first n nodes

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

**usercutfreq** *(integer)*: determines the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10
usercutinterval (integer): determines the interval when to apply the multiplier for the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 100

usercutmult (integer): determines the multiplier for the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 2

usercutnewint (boolean): calls the cut generator if the solver found a new integer feasible solution

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 1

usercutpool (boolean): Indicator to use user cuts

Default: 0

usergdxin (string): the name of the GDX file read back into Cplex

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchin.gdx

usergdxname (string): the name of the GDX file exported from the solver with the solution at the node

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchout.gdx

usergdxnameinc (string): the name of the GDX file exported from the solver with the incumbent solution

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchout_i.gdx

usergdxprefix (string): prefixes usergdxin, usergdxname, and usergdxnameinc

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

usergdxsol (string): the name of the GDX file exported by Cplex to store the solution of extra columns

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchsol.gdx

userheurcall (string): the GAMS command line to call the heuristic

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userheurfirst (integer): calls the heuristic for the first n nodes
More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

**userheurfreq (integer):** determines the frequency of the heuristic model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

**userheurinterval (integer):** determines the interval when to apply the multiplier for the frequency of the heuristic model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 100

**userheurmult (integer):** determines the multiplier for the frequency of the heuristic model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 2

**userheurnewint (boolean):** calls the heuristic if the solver found a new integer feasible solution

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 1

**userheurobjfirst (integer):** Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 0

**userincbcall (string):** the GAMS command line to call the incumbent checking program

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

**userincbicall (string):** the GAMS command line to call the incumbent reporting program

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

**userjobid (string):** postfixes If, o on call adds –userjobid to the call. Postfixes gdxname, gdxnameinc and gdxin

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

**userkeep (boolean):** calls gamskeep instead of gams

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 0

**userlazyconcall (string):** the GAMS command line to call the lazy constraint generator

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Note: There is no guarantee that CPLEX will use all of the added violated lazy constraints provided due to technical and/or efficiency reasons. It may thus happen that a later candidate solution violates previously provided lazy constraints. In this case consider passing the constraint again.

**varsel (integer):** variable selection strategy at each node

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Branch on variable with minimum infeasibility. This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.</td>
</tr>
<tr>
<td>0</td>
<td>Branch variable automatically selected</td>
</tr>
<tr>
<td>1</td>
<td>Branch on variable with maximum infeasibility. This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.</td>
</tr>
<tr>
<td>2</td>
<td>Branch based on pseudo costs. Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.</td>
</tr>
<tr>
<td>3</td>
<td>Strong Branching. This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.</td>
</tr>
<tr>
<td>4</td>
<td>Branch based on pseudo reduced costs</td>
</tr>
</tbody>
</table>

**warninglimit** *(integer)*: determines how many times warnings of a specific type *(datacheck=2)* will be displayed \(\leftarrow\)

By default, when modeling assistance is turned on via the *data consistency checking* parameter, CPLEX will display 10 warnings for a given modeling issue and then omit the rest. This parameter controls this limit and allows the user to display all of the warnings if desired. In order to see all warnings change the value to its negative.

Default: 10

**workdir** *(string)*: directory for working files \(\leftarrow\)

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

Default: current or project directory

**workeralgorithm** *(integer)*: set method for optimizing benders subproblems \(\leftarrow\)

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Primal Simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual Simplex</td>
</tr>
<tr>
<td>3</td>
<td>Network Simplex</td>
</tr>
<tr>
<td>4</td>
<td>Barrier</td>
</tr>
<tr>
<td>5</td>
<td>Sifting</td>
</tr>
</tbody>
</table>

**workmem** *(real)*: memory available for working storage \(\leftarrow\)

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter *WorkDir*.

Default: 2048.0
writeannotation (string): produce a Cplex annotation file

writebas (string): produce a Cplex basis file

Write a basis file.

writeflt (string): produce a Cplex solution pool filter file

Write the diversity filter to a Cplex FLT file.

writelp (string): produce a Cplex LP file

Write a file in Cplex LP format.

writemps (string): produce a Cplex MPS file

Write an MPS problem file.

writemst (string): produce a Cplex mst file

Write a Cplex MST (containing the MIP start) file.

writeord (string): produce a Cplex ord file

Write a Cplex ORD (containing priority and branch direction information) file.

writeparam (string): produce a Cplex parameter file with all active options

Write a Cplex parameter (containing all modified Cplex options) file.

writepre (string): produce a Cplex LP/MPS/SAV file of the presolved problem

Synonym: writepremps

Write a Cplex LP, MPS, or SAV file of the presolved problem. The file extension determines the problem format. For example, `WritePre presolved.lp` creates a file `presolved.lp` in Cplex LP format.

writeprob (string): produce a Cplex problem file and infers the type from the extension

Write a problem file in a format inferred from the extension. Possible formats are

- **SAV**: Binary matrix and basis file
- **MPS**: MPS format
- **LP**: CPLEX LP format with names modified to conform to LP format
- **REW**: MPS format, with all names changed to generic names
- **RLP**: LP format, with all names changed to generic names
- **ALP**: LP format, with generic name of each variable, type of each variable, bound of each variable If the file name ends with `.bz2` or `.gz`, a compressed file is written.

writesav (string): produce a Cplex binary problem file

Write a binary problem file.

zerohalfcuts (integer): zero-half cuts

Decides whether or not to generate zero-half cuts for the problem. The value 0, the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If the dual bound of your model does not make sufficient progress, consider setting this parameter to 2 to generate zero-half cuts more aggressively.

Default: 0
5.11.9 Setting up a GAMS/Cplex-Link license

The GAMS/Cplex-Link license requires that you have a valid license agreement with IBM for use of the current version of the Cplex library.

To add the Cplex-Link to your GAMS license, please write to sales@gams.com to confirm that you have such a license agreement and that your use of the GAMS solver will comply with the terms of that license agreement. Please reference your GAMS License ID, DCxxxx.

5.12 Deterministic Equivalent (DE)

Author
Martha Loewe

5.12.1 Introduction

DE is a solver for stochastic programs modeled with GAMS Extended Mathematical Programming for Stochastic Programming (EMP SP). DE can solve multi-stage LP, MIP, QCP, NLP and MINLP stochastic programming models. For details about EMP SP and the syntax to modify an existing GAMS model to be an stochastic programming model in GAMS EMP SP see Stochastic Programming. A list of DE solver options is given at the end of this document.

Stochastic programs are mathematical programs that include data that is not known with certainty, but is approximated by probability distributions. The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two variables and let the set of all realizations of the unknown data be given by $\Omega$, $\Omega = \{\omega_1, \ldots, \omega_S\} \subseteq \mathbb{R}^r$, where $r$ is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

$$
\begin{align}
\text{Min}_x \quad & z = c^T x + \mathbb{E}[Q(x, \omega)] \\
\text{s.t.} \quad & A x = b, \quad x \geq 0,
\end{align}
$$

where $Q(x, \omega) = \begin{align}
\text{Min}_y \\
\text{s.t.} \\
T_\omega x + W_\omega y(\omega) = h_\omega, \quad y(\omega) \geq 0, \quad \forall \omega \in \Omega.
\end{align}$

The first two lines define the first-stage problem and the last two lines define the second-stage problem. In the first stage, $x$ is the decision variable, $c^T$ represents the cost coefficients of the objective function and $\mathbb{E}[Q(x, \omega)]$ denotes the expected value of the optimal solution of the second stage problem. In addition, $A$ denotes the coefficients and $b$ the right-hand side of the first stage constraints. In the second stage, $y$ is the decision variable, $T$ represents the transition matrix, $W$ the recourse matrix (cost of recourse) and $h$
the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on $\omega$. The objective variable $z$ is also a random variable, since it is a function of $\omega$. As a random variable cannot be optimized, DE automatically optimizes the expected value of the objective variable $z$. DE is also able to optimize other risk measures, as will be discussed later.

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then the values that the uncertain parameters take will become known. Given the fixed solution of the first stage and the new data, in the second stage recourse action can be taken and the optimal solution determined. Each possible realization of the uncertain data is represented by an $\omega_s \in \Omega$ and is called a scenario. The objective is to find a feasible solution $x$ that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the deterministic equivalent. Assume that the uncertain parameters follow a (finite) discrete distribution and that each scenario $\omega_s$ occurs with probability $P(\omega_s) = p_s$ for all $s = 1, \ldots, S$ and $\sum_s p_s = 1$. So $E[Q(x, \omega)] = \sum_s p_s q^T y_s$, where $y_s$ denotes the optimal second-stage decision for the scenario $\omega_s$. Then the deterministic equivalent can be expressed as follows:

$$
\begin{align*}
& c^T x + p_1 q^T y_1 + p_2 q^T y_2 + \ldots + p_S q^T y_S \\
& \text{s.t.} \\
& Ax + W_1 y_1 = b \\
& T_2 x + W_2 y_2 = h_2 \\
& \vdots \\
& T_S x + W_S y_S = h_S \\
& x \in \mathbb{R}^n, \quad y_1 \in \mathbb{R}^m, \quad y_2 \in \mathbb{R}^m, \ldots, \quad y_S \in \mathbb{R}^m
\end{align*}
$$

(5.7)

Note that for stochastic linear programs the deterministic equivalent is just a very large linear program.

The solver DE automatically generates the deterministic equivalent of a stochastic program and then hands it over for solution to a solver, termed the subsolver. The subsolver is the default solver for the type of model to be solved. The user may choose another subsolver with the option subsolver.

If modelers want to use DE, they may either add the option

```
option emp = de;
```

before the solve statement in the model or they may choose the solver DE on the command line, e.g.

```
gams mymodel.gms emp=de
```

### 5.12.2 Random Variables

Random variables and their distributions and stages determine the number of scenarios in the model and thus the number of equations that are generated for the deterministic equivalent. If there are two or more random variables in one stage they may be jointly distributed or not.

Assume that there is a two-stage stochastic model $\mathcal{M}_2$ that has two random variables $a$ and $b$ in the second stage. The random variables both follow a discrete distribution, the uncertain data for $a$ is given by $\Omega_a$ and the uncertain data for $b$ is given by $\Omega_b$. If the random variables are not jointly distributed, then $|\Omega_a| \times |\Omega_b|$ scenarios are generated. If the random variables are jointly distributed, then we need to have $|\Omega_a| = |\Omega_b|$ and the number of scenarios is $|\Omega_a|$. 
In addition to two-stage stochastic models, DE can also solve multi-stage stochastic models. Let $M_4$ be a stochastic model with four stages. In each of the stages after the first stage new data is revealed that was unknown in the previous stages. Assume that in each stage there is just one random variable. Let $\Omega_I$ denote the uncertain data in the second stage and $\Omega_{II}$ and $\Omega_{III}$ represent the uncertain data of the third and fourth stage respectively. Let $|\Omega_I| = n_1$, $|\Omega_{II}| = n_2$ and $|\Omega_{III}| = n_3$. DE builds the deterministic equivalent of this model by recursively generating a scenario tree. In the second stage the number of scenarios is $n_1$, in the third stage there are $n_1 \times n_2$ scenarios and in the fourth stage there are $n_1 \times n_2 \times n_3$ scenarios. Note that the number of scenarios in the model equals the number of scenarios in the last stage. However, the number of equations generated for the deterministic equivalent is the sum of the scenarios in each stage, so in this example it is $n_1 + n_1 \times n_2 + n_1 \times n_2 \times n_3 = n_1 \times (1 + n_2 \times (1 + n_3))$.

The following figure illustrates the scenario tree of a stochastic model with four stages.

![Scenario Tree Illustration](image)

Note that the structure of the scenario tree is limited: each node in a stage has exactly the same number of arcs.

If there is more than one random variable per stage, then the number of scenarios increases accordingly. Note that there is no absolute limit to the number of stages and random variables, but the model can become very large very quickly as the number of stages and random variables increases. Note further that the random variables are stagewise independent. The realization of the random variables in one stage does not influence the realization of the random variables in a later stage.

### 5.12.3 Sampling Procedures

DE can solve only stochastic programs with random variables that follow discrete probability distributions. If the random data follows a continuous probability distribution the set $\Omega$ has an infinite number of elements. Sampling is a process that generates a finite approximation $\Omega_N$ to $\Omega$. The size of $\Omega_N$ is $N$ and each element of $\Omega_N$ has the same probability $p = \frac{1}{N}$. Thus continuous probability distributions are approximated by discrete distributions. EMP SP provides the keyword `sample` to facilitate sampling. An example follows.

```
randvar d normal 45 10
sample d 100
```
These two lines are a portion of the `emp.info` file. In the first line `d` is defined as a random variable that follows a Normal distribution with mean 45 and standard deviation 10. In the second line it is specified that the continuous distribution is approximated by a sample of size 100. Note that the user determines the sample size \( N \). Note further that the use of the keyword `sample` is mandatory for random variables with continuous distributions. Otherwise the following error message will appear:

*** Only random variables with sampled continuous distributions supported.

Observe that behind the scenes the sampling is performed by the LINDO system. So the modeler needs a LINDO licence for sample sizes larger than 10 (smaller sample sizes are included in the demo version). The LINDO system offers three variance reduction algorithms: the Antithetic algorithm, the Latin Square algorithm and the Monte Carlo algorithm. They may be enabled using either the option `svr_ls_antithetic` or `svr_ls_latinsquare` or `svr_ls_montecarlo` respectively.

Alternatively, the modeler may chose to generate a sample of the distribution first and then enter the sample as a discrete distribution in the `emp.info` file. There are two stochastic libraries that can be used for this procedure: the GAMS Stochastic Library and the LINDO Sampling Library. An example and details about the procedure using the LINDO library are given in section Sampling in the EMP manual.

### 5.12.4 The Expected Value Problem

Consider the special case where the sample size is 1 and the "sampled" value equals the expected value of the random variable. This entails that the random variable is replaced by its expected value. The resulting model is deterministic and is called the Expected Value Program. DE facilitates solving the Expected Value Problem through the option `solveEVProb`. If this option is specified in the option file (see example below) the Expected Value Problem is solved after the original stochastic model and the solution is reported. In addition, DE offers the option to choose a separate subsolver for the Expected Value Problem, the option `evsubsolver`.

In the following example the stochastic model is solved with the subsolver Conopt and the Expected Value Problem is solved with the subsolver Minos.

```plaintext
$onecho > de.opt
subsolver conopt
solveEVProb
evsubsolver minos
$offecho
mymodel.optfile=1;
solve mymodel min obj using emp scenario dict;
```

Note that the option `solveEVProb` is not defined for models with chance constraints and models featuring VaR and CVaR. These model types are discussed in the next section.

### 5.12.5 What types of models can DE handle?

As mentioned earlier, DE can solve two-stage and multi-stage stochastic linear, quadratic and nonlinear models and the mixed-integer versions of all these models. There are two further classes of stochastic models that DE is equipped to solve: stochastic programs with chance constraints and stochastic programs with other risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR). CVaR is also called Expected Shortfall.
In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints to be violated with a certain probability. Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

\[
\begin{align*}
\text{Min} & \quad c^T x \\
\text{s.t.} & \quad P(Ax \leq b) \geq p \\
& \quad x \geq 0,
\end{align*}
\]  \hspace{1cm} (5.8)

where \( x \in \mathbb{R}^n \) is the decision variable and \( c^T \) denotes the coefficients of the objective function, \( A \in \mathbb{R}^{m \times n} \) is a random matrix and represents the coefficients and \( b \in \mathbb{R}^m \) is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability \( \epsilon = 1 - p \), where \( 0 < p \leq 1 \).

DE offers three reformulation options to solve stochastic programs with chance constraints: a reformulation using a mixed-integer program with big \( M \) notation, a convex hull reformulation and the use of indicator variables and indicator constraints. Note that the default is the MIP reformulation (the default value of \( M \) is set to 10000 and can be customized). With the option ccreform the reformulation method can be determined by the user.

For details on single and joint chance constraints and examples how to use the option ccreform, see the section Chance Constraints with EMP in the EMP manual. Note that random variables in programs with chance constraints may follow discrete or continuous probability distributions. Note further that there are no stages in stochastic programs with chance constraints.

As mentioned earlier, DE automatically optimizes the expected value of the objective function variable. DE also supports other risk measures, such as VaR and CVaR. In mathematical terms, DE is able to solve the following:

\[
\text{Min}_x \quad \mathcal{R}(z),
\]  \hspace{1cm} (5.9)

where \( z \) represents the objective function variable and \( \mathcal{R} \) denotes a risk measure such as the Expected Value, VaR and CVaR or a combination of risk measures (e.g. the weighted sum of Expected Value and CVaR). In addition, the modeler can also choose to trade off risk measures. For more details about risk measures see section Risk Measures with EMP in the EMP manual.

### 5.12.6 Reformulation Techniques

In this section some details are given about the reformulations that DE is performing behind the scenes.

#### 5.12.6.1 Simple two-stage stochastic model

As mentioned previously, DE converts a stochastic model into its deterministic equivalent. Using the model nbsimple.gms from the GAMS EMP model library as an example, we show how exactly the deterministic equivalent is built. Note that this model is also discussed in detail in the section A Simple Example: The News Vendor Problem of the EMP manual.

The model is given by the following equations:

\[
\begin{align*}
\text{Max}_x \quad z(x) = -cx + \mathbb{E}[Q(x, \omega)], & \quad x \geq 0,
\end{align*}
\]  \hspace{1cm} (5.10)
where
\[ Q(x, \omega) = \max_{s, i, l} vs(\omega) - hi(\omega) - pl(\omega) \]
subject to
\[ x - s(\omega) - i(\omega) = 0 \]
\[ s(\omega) + l(\omega) = d(\omega) \]
\[ s(\omega), i(\omega), l(\omega) \geq 0 \quad \forall \omega \in \Omega. \]

Here \( c, v, h \) and \( p \) are some given parameters, the first-stage decision variable is \( x \), the second-stage decision variables are \( s, i \) and \( l \) and the uncertain data is represented by the random variable \( d(\omega) \). We assume that the annotations specify that \( d \) follows a Normal distribution and the sample size to approximate this continuous distribution is 4 (where all scenarios are equally likely, so \( p(s) = 0.25 \)).

First, DE draws 4 values from the Normal distribution specified in the annotations. These 4 values are the basis for the 4 scenarios that are generated. For each scenario, a separate equation for the two second-stage constraints is built (that are 8 equations). In the process, for each of the three decision variables of the second stage 4 variables are generated since the second-stage decision variables may take different values for each scenario. Next, an equation for the profit for each scenario is generated (4 equations) and finally, the equation to compute the average profit is built. So there are a total of 13 equations. They are given here:

\[
\begin{align*}
& x - s_1 - i_1 = 0 \\
& x - s_2 - i_2 = 0 \\
& x - s_3 - i_3 = 0 \\
& x - s_4 - i_4 = 0 \\
& s_1 + l_1 = d_1 \\
& s_2 + l_2 = d_2 \\
& s_3 + l_3 = d_3 \\
& s_4 + l_4 = d_4 \\
& z_1 = -cx + vs_1 - hi_1 - pl_1 \\
& z_2 = -cx + vs_2 - hi_2 - pl_2 \\
& z_3 = -cx + vs_3 - hi_3 - pl_3 \\
& z_4 = -cx + vs_4 - hi_4 - pl_4 \\
& z = 0.25z_1 + 0.25z_2 + 0.25z_3 + 0.25z_4
\end{align*}
\]

Note that there is only one realization of the first-stage decision variable \( x \) while for the second-stage decision variables there as many realizations as there are scenarios.

### 5.12.6.2 A multi-stage stochastic model

DE recursively builds a scenario tree to generate the deterministic equivalent of multi-stage models. We take the model inventory from the EMP manual to demonstrate how this is done. The 4-stage model is given by the following equations:

\[
\begin{align*}
\max_{y_1, i_1} & \quad z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[Q((y_1, i_1), \omega_1)] + \mathbb{E}[Q((y_1, i_1), \omega_{II})] + \mathbb{E}[Q((y_1, i_1), \omega_{III})] \\
\text{s.t.} & \quad i_1 = y_1 \\
& \quad y_1, i_1 \geq 0,
\end{align*}
\]

where
\[
Q((y_1, i_1), \omega_1^t) = \max_{s_t, y_t, i_t} \beta s_t(\omega_1^t) - \alpha y_t(\omega_1^t) - \delta i_t(\omega_1^t) \\
\text{s.t.} & \quad i_{t-1} + y_t = s_t + i_t \\
& \quad s_t \leq i_{t-1} \\
& \quad s_t \leq d_t(\omega_1^t) \\
& \quad i_t \leq \kappa \\
& \quad s_t, y_t, i_t \geq 0 \quad \text{for} \quad t = 2, 3, 4.
\]
Here \( y_t \) and \( i_t \) are the first-stage decision variables and \( y_t, i_t \) and \( s_t \) are the decision variables of the later stages; \( \alpha, \beta, \gamma, \delta \) and \( \kappa \) denote fixed parameters. The uncertain data is represented by the random variable \( d_t(\omega_\zeta) \), where \( \omega_\zeta \in \Omega_\zeta \) and \( \zeta = I, II, III \). Note that \( s_t, y_t \) and \( i_t \) depend on the realization of \( d_t \). Now assume that \( d_t \) follows a continuous distribution and the sample size to approximate this distribution is just 3 for stages 2 to 4. Note that in multi-stage models the sample size may vary for each stage. Note further that for all practical purposes this sample size is too small, it was chosen here for reasons of clarity of exposition. As usual, the distribution(s) and sample size(s) are specified in the annotations. Observe that the last constraint \( i_t \leq \kappa \) is implemented as a bound on the variable \( i_t \), not as an equation. So, in addition to the objective equations, there is one equation in the first stage and three equations in each of the later stages.

This model is reformulated in the following way. First, an equation for the first stage constraint is generated. Secondly, three realizations of \( d_2 \) are chosen through a sampling procedure, so there are three scenarios at stage 2. For each of these three realizations of the random variable the equations for the second stage are generated (9 equations). Thirdly, three realizations of \( d_3 \) are sampled for stage 3. Each of these three realizations may follow one of the three scenarios of the second stage, so there are nine scenarios in stage 3 and hence \( 9 \times 3 = 27 \) equations are generated for stage 3. The procedure is repeated in stage 4 where there are 27 scenarios and \( 27 \times 3 = 81 \) equations. Next, the value of the objective variable for each scenario is computed by generating 27 equations for \( z_s \) and finally, an equation to compute \( z \) is built, where \( z = \sum_s p(s)z_s \).

5.12.6.3 Chance Constraints

Models with chance constraints are reformulated as mixed integer problems by default. This reformulation is discussed in detail in the section Chance Constraints with EMP in the EMP manual. Reformulations using a convex hull or indicator variables and indicator constraints are also possible. The reformulation method may be determined by the user with the option ccreform.

5.12.6.4 Computing VaR

Models involving VaR are reformulated using mixed integer programs with big \( M \) notation. By the example model portfolio.gms we demonstrate how this is done. This model is discussed in the section Value at Risk (VaR) in the EMP manual. The model is given by the following equations:

\[
\begin{align*}
\text{Max} & \quad \text{VaR}_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j(\omega) \quad \forall \omega \in \Omega \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]

where \( \text{VaR}_\theta \) is the VaR at the lower \( \theta \)th percentile, the variable \( R \) is the return (and is a function of the random variable \( v_j(\omega) \)), \( w_j \) the weight associated with each asset \( j \) and \( v_j(\omega) \) is the (random variable) return of each asset \( j \). The weights can also be interpreted as proportions of the amount to be invested, their sum must be 1. Note that \( w_j \) is the decision variable in this problem.

Let \( y(s) \) be a binary variable that acts as an indicator variable. For each scenario, it equals 1 if the return \( R \geq \text{VaR}_\theta[R] \) and it is 0 otherwise. Then the model is reformulated to the following MIP:

\[
\begin{align*}
\text{Max} & \quad \text{VaR}_\theta[R] \\
\text{s.t} & \quad r(s) = \sum_j w_j v_j(s) \\
& \quad r(s) \geq \text{VaR}_\theta[R] - M(1 - y(s)) \quad \forall s = 1, \ldots, S \\
& \quad cc = 1 - \sum_s p(s)y(s) \\
& \quad \sum_j w_j = 1 \\
& \quad 0 \leq cc \leq \theta \\
& \quad w_j \geq 0.
\end{align*}
\]
Here \( r(s) \) denotes the return per scenario and \( cc \) is a variable that represents the probability of violations (a violation occurs if in a scenario the return is smaller than \( VaR_\theta[R] \)). As usual, \( p(s) \) is the probability that a scenario occurs and \( M \) is the big \( M \). The first three constraints together with the bounds on \( cc \) ensure that the probability of violations is smaller than or equal to \( \theta \).

Note that the default value for big \( M \) for models with VaR is 1000 (and it is different from the default value for big \( M \) for chance constraints). It may be customized with the option varbigm.

5.12.6.5 Computing CVaR

Models with CVaR are reformulated by converting the annotations into new variables and equations. Using the model portfolio.gms as an example we demonstrate how this is done. This model features in the GAMS EMP library and is also discussed in the section Conditional Value at Risk (CVaR) in the EMP manual. The model is given by:

\[
\begin{align*}
\text{Max} & \quad CVaR_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j(\omega) \quad \forall \omega \in \Omega \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]  

(5.14)

where \( CVaR_\theta \) is the CVaR at the left tail of the distribution at the confidence level \( \theta \) and all other variables are as defined above. Note that again, \( w_j \) is the decision variable in this problem.

DE reformulates this model by introducing three new variables and three associated equations. Let \( r(s) \) denote the return per scenario, \( t \) the target return to be achieved with probability approximately \( 1 - \theta \) and \( d(s) \), \( d(s) \geq 0 \), the lower deviation from \( t \) per scenario. The equations follow.

\[
\begin{align*}
\text{Max} & \quad CVaR_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j(s) \\
& \quad d(s) \geq t - r(s) \\
& \quad CVaR_\theta[R] = t - \frac{1}{\theta} \sum p(s) d(s)
\end{align*}
\]  

(5.15)

Thus the model is reformulated to an LP and may be solved by an appropriate solver. It is easy to see that \( t \) equals \( VaR_\theta[R] \).

5.12.7 Logfile

The logfile gives much information about the solver progress. The following is the DE log output from running the nbdiscjoint model from the GAMS EMP Model Library:

--- Starting compilation
--- nbdisjoint.gms(74) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- nbdisjoint.gms(72) 4 Mb
---
--- collecting and writing gdx file
---
--- Generating EMP model nb
--- nbdisjoint.gms(72) 6 Mb
--- 5 rows 9 columns 19 non-zeroes
--- 6 nl-code 2 nl-non-zeroes
First, the model in scalar form is generated. It has 5 equations, so the model statistics report 5 rows. Further, it has 6 positive variables, one free variable and 2 random variables, so there are 9 columns. A matrix form representation of the model with the variables as columns and the equations as rows has 19 non-zero entries. Next, DE generates the deterministic equivalent. The statistics for the deterministic equivalent indicate that there are 26 rows, 50 columns and 116 non-zeroes. The stochastic model has 6 scenarios, so for each second-stage equation there are 6 equations in the deterministic equivalent (i.e. a total of 24 equations). In addition, there is one first-stage equation and one equation to compute the expected value of the objective variable, which brings the sum total to 26 equations or rows. Similarly, for each second-stage variable in the stochastic model there are 6 variables in the deterministic equivalent (i.e. a total of 48 variables). In addition, there is the first-stage decision variable and a variable for the expected value of the objective adding up to a total of 50 variables or columns.

Note that the model type is identified as an LP and thus the default LP solver is invoked. In this case the subsolver is Cplex and most of the remainder of the logfile is output from the subsolver. At the end the mapping specified in the dictionary is performed.
The logfile of solving stochastic models with VaR or CVaR is similar. It reports that the model type is being determined, the deterministic equivalent built and then handed over to the appropriate subsolver to be solved. However, when solving stochastic programs with chance constraints there is much more happening behind the scenes. DE automatically hands over the model to the subsolver JAMS. By default, JAMS reformulates the model and generates a scalar version of it. The scalar version of the reformulated model is then handed back to the GAMS system to be solved by an appropriate subsolver. The logfile reports the progress of the subsolver until the model is solved. Then the solution from the GAMS subsolver is handed to JAMS and a list of disjunctions and their activity level is reported. (The equations in the scalar version of the reformulated model that may be left unsatisfied are called disjunctions. Active disjunctions refer to equations that are satisfied and not active disjunctions refer to equations that are not satisfied.)

Parts of the DE logfile from running the simplechance model from the GAMS EMP Model Library are given below. These parts of the logfile demonstrate the solution process for stochastic models with chance constraints that was just described.

--- Generating EMP model sc
--- simplechance.gms(79) 6 Mb
--- 3 rows 5 columns 9 non-zeroes
--- 9 nl-code 4 nl-non-zeroes
--- simplechance.gms(79) 4 Mb
--- Executing DE: elapsed 0:00:00.024
DE 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
--- Reset Solvelink = 2

JAMS 1.0 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X

JAMS - Solver for Extended Mathematical Programs (EMP)
-----------------------------------------------

--- Using Option File
Reading parameter(s) from "jams.159"
>> EMPInfoFile .../Models/225a/jamsinfo.dat
>> SubSolver CPLEX
Finished reading from "jams.159"

--- EMP Summary
Logical Constraints = 0
Disjunctions = 7
Adjusted Constraint = 0
Flipped Constraints = 0
Dual Variable Maps = 0
Dual Equation Maps = 0
VI Functions = 0
Equilibrium Agent = 0
Bilevel Followers = 0

*** Warning 7 of 7 BigM disjunctions use DE's default for bigM (10000)

--- The model .../Models/225a/emp.dat will be solved by GAMS
---
--- Job emp.dat Start 12/06/15 18:43:18 24.5.1 r54187 DEX-DEG x86 64bit/MacOS X
GAMS 24.5.1 Copyright (C) 1987-2015 GAMS Development. All rights reserved
...
--- Generating MIP model m
--- emp.dat(68) 3 Mb  
--- 10 rows 19 columns 40 non-zeroes  
--- 7 discrete-columns  
--- Executing CPLEX: elapsed 0:00:00.016

IBM ILOG CPLEX 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X  
Cplex 12.6.2.0

Reading data...  
Starting Cplex...  
....

MIP Solution: 4.750000 (11 iterations, 0 nodes)  
Final Solve: 4.750000 (2 iterations)

--- Reading solution for model m

*** Status: Normal completion
--- Job emp.dat Stop 12/06/15 18:43:18 elapsed 0:00:00.299
--- Returning from GAMS step
---

--- Disjunction Summary
  Disjunction 1 is not active
  Disjunction 2 is active
  Disjunction 3 is active
  Disjunction 4 is active
  Disjunction 5 is not active
  Disjunction 6 is active
  Disjunction 7 is active
---

--- Restarting execution
--- simplechance.gms(79) 2 Mb
--- Reading solution for model sc
--- simplechance.gms(79) 3 Mb
---

--- scattering and reading gdx file
---
--- randvar Id = om1 maps to s_om1
--- simplechance.gms(79) 3 Mb
--- randvar Id = om2 maps to s_om2
--- level Id = x1 maps to x1_l
--- marginal Id = x1 maps to x1_m
--- level Id = x2 maps to x2_l
--- level Id = e1 maps to e1_l
--- level Id = e2 maps to e2_l
--- simplechance.gms(79) 4 Mb
--- Scatter finished in 2 ms
--- simplechance.gms(81) 4 Mb
*** Status: Normal completion

5.12.8 Summary of DE Options

5.12.8.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlationtype</td>
<td>Sample correlation type</td>
<td>0</td>
</tr>
<tr>
<td>subsolver</td>
<td>Subsolver to run</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>subsolveropt</td>
<td>Optfile value to pass to the subsolver</td>
<td>1</td>
</tr>
<tr>
<td>svr_ls_antithetic</td>
<td>Sample variance reduction map to Lindo Antithetic algorithm</td>
<td></td>
</tr>
<tr>
<td>svr_ls_latin square</td>
<td>Sample variance reduction map to Lindo Latin Square algorithm</td>
<td></td>
</tr>
<tr>
<td>svr_ls_montecarlo</td>
<td>Sample variance reduction map to Lindo Montecarlo algorithm</td>
<td></td>
</tr>
</tbody>
</table>

### 5.12.8.2 Options for chance constraint models

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccreform</td>
<td>Reformulation option passed to JAMS</td>
<td>bigM</td>
</tr>
<tr>
<td>jamsopt</td>
<td>JAMS option file</td>
<td></td>
</tr>
</tbody>
</table>

### 5.12.8.3 Options for recourse models

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>evsubsolver</td>
<td>Subsolver to run on expected value problem</td>
<td></td>
</tr>
<tr>
<td>evsubsolveropt</td>
<td>Optfile value to pass to the subsolver for expected value problem</td>
<td>1</td>
</tr>
<tr>
<td>maxnodes</td>
<td>Tree size limit</td>
<td>100000</td>
</tr>
<tr>
<td>solveEVProb</td>
<td>Solve and report the expected value solution</td>
<td>0</td>
</tr>
<tr>
<td>varbigm</td>
<td>Big M for Value at Risk reformulation</td>
<td>1000.0</td>
</tr>
</tbody>
</table>

### 5.12.9 Detailed Descriptions of DE Options

**ccreform (string):** Reformulation option passed to JAMS

This option determines how to formulate the indicator constraints of the chance constraints in the deterministic equivalent. The model is passed to JAMS for the actual reformulation and solution with a subsolver. The possible reformulation options are `bigM [big eps threshold]`, `chull [big eps]`, and `indic`. The `indic` setting only works with solver that under stand the indicator syntax in a GAMS option file.

Default: `bigM`

**correlationtype (integer):** Sample correlation type

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pearson</td>
</tr>
<tr>
<td>1</td>
<td>Kendall</td>
</tr>
<tr>
<td>2</td>
<td>Spearman</td>
</tr>
</tbody>
</table>

**evsubsolver (string):** Subsolver to run on expected value problem
**5.13 DECIS**

Author

Gerd Infanger; Vienna University of Technology; Stanford University

---

1Copyright ©1989 – 1999 by Gerd Infanger. All rights reserved. The GAMS/DECIS User’s Guide is copyrighted and all rights are reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Gerd Infanger. The DECIS software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement. The DECIS software can be licensed through Infanger Investment Technology, LLC or through Gams Development Corporation.
5.13.1 DECIS

5.13.1.1 Introduction

DECIS is a system for solving large-scale stochastic programs, i.e. programs that include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to be known by their probability distribution. It employs Benders decomposition and advanced Monte Carlo sampling techniques. DECIS includes a variety of solution strategies, such as solving the universe problem, the expected value problem, Monte Carlo sampling within the Benders decomposition algorithm, and Monte Carlo pre-sampling. When using Monte Carlo sampling the user has the option of employing crude Monte Carlo without variance reduction techniques, or using as variance reduction techniques importance sampling or control variates, based on either an additive or a multiplicative approximation function. Pre-sampling is limited to using crude Monte Carlo only.

For solving linear and nonlinear programs (master and subproblems arising from the decomposition) DECIS interfaces with MINOS or CPLEX. MINOS, see Murtagh and Saunders (1983) [142], is a state-of-the-art solver for large-scale linear and nonlinear programs, and CPLEX is one of the fastest linear programming solvers available.

For details about the DECIS system consult the DECIS User’s Guide see Infanger (1997) [100]. It includes a comprehensive mathematical description of the methods used by DECIS. In this Guide we concentrate on how to use DECIS directly from GAMS, and especially on how to model stochastic programs using the GAMS/DECIS interface. First, however, in section What DECIS Can Do we give a brief description of what DECIS can do and what solution strategies it uses. This description has been adapted from the DECIS User’s Guide. In section GAMS/DECIS we discuss in detail how to set up a stochastic problem using GAMS/DECIS and give a description of the parameter setting and outputs obtained. In Appendix A - GAMS/DECIS Illustrative Examples we show the GAMS/DECIS formulation of two illustrative examples (APL1P and APL1PC) discussed in the DECIS User’s Guide. A list of DECIS error messages are represented in Appendix B - Error Messages.

In addition to using the syntax explained in this document, it is also possible to use the Extended Mathematical Programming (EMP) framework to define a stochastic program for GAMS/DECIS. This possibility was introduced with GAMS 23.8 and allows one to solve a stochastic program not only with GAMS/DECIS but also with GAMS/DE and GAMS/Lindo. Further information about this can be found in the chapter Stochastic Programming (SP) with EMP.

5.13.1.2 What DECIS Can Do

DECIS solves two-stage stochastic linear programs with recourse:

\[
\min \quad z = cx + Ef^\omega y^\omega \\
\text{s.t.} \quad Ax = b \\
\quad -B^\omega x + D^\omega y^\omega = d^\omega \\
\quad x, y^\omega \geq 0, \quad \omega \in \Omega.
\]

where \( x \) denotes the first-stage, \( y^\omega \) the second-stage decision variables, \( c \) represents the first-stage and \( f^\omega \) the second-stage objective coefficients, \( A, b \) represent the coefficients and right hand sides of the first-stage constraints, and \( B^\omega, D^\omega, d^\omega \) represent the parameters of the second-stage constraints, where the transition matrix \( B^\omega \) couples the two stages. In the literature \( D^\omega \) is often referred to as the technology matrix or recourse matrix. The first stage parameters are known with certainty. The second stage parameters are
random parameters that assume outcomes labeled $\omega$ with probability $p(\omega)$, where $\Omega$ denotes the set of all possible outcome labels.

At the time the first-stage decision $x$ has to be made, the second-stage parameters are only known by their probability distribution of possible outcomes. Later, after $x$ has already been determined, an actual outcome of the second-stage parameters will become known, and the second-stage decision $y^\omega$ is made based on knowledge of the actual outcome $\omega$. The objective is to find a feasible decision $x$ that minimizes the total expected costs, the sum of first-stage costs and expected second-stage costs.

For discrete distributions of the random parameters, the stochastic linear program can be represented by the corresponding equivalent deterministic linear program:

$$
\min z = cx + p_1fy_1 + p_2fy_2 + \cdots + p_Wfy_W \\
\text{s/t } Ax - B_1x + Dy_1 - B_2x + Dy_2 - \cdots - B_Wx + Dy_W = b \\
\quad x, y_1, y_2, \ldots, y_W \geq 0,
$$

which contains all possible outcomes $\omega \in \Omega$. Note that for practical problems $W$ is very large, e.g., a typical number could be $10^{20}$, and the resulting equivalent deterministic linear problem is too large to be solved directly.

In order to see the two-stage nature of the underlying decision making process the following representation is also often used:

$$
\min cx + E z^\omega(x) \\
\text{s/t } Ax = b \\
x \geq 0,
$$

where

$$
z^\omega(x) = \min f^\omega y^\omega \\
D^\omega y^\omega = d^\omega + B^\omega x \\
y^\omega \geq 0, \omega \in \Omega = \{1, 2, \ldots, W\}.
$$

DECIS employs different strategies to solve two-stage stochastic linear programs. It computes an exact optimal solution to the problem or approximates the true optimal solution very closely and gives a confidence interval within which the true optimal objective lies with, say, 95% confidence.

5.13.1.3 Representing Uncertainty

It is favorable to represent the uncertain second-stage parameters in a structure. Using $V = (V_1, \ldots, V_h)$ an $h$-dimensional independent random vector parameter that assumes outcomes $v^\omega = (v_1, \ldots, v_h)^\omega$ with probability $p^\omega = p(v^\omega)$, we represent the uncertain second-stage parameters of the problem as functions of the independent random parameter $V$: 
Each component $V_i$ has outcomes $v_{i\omega}^i$, $\omega_i \in \Omega_i$, where $\omega_i$ labels a possible outcome of component $i$, and $\Omega_i$ represents the set of all possible outcomes of component $i$. An outcome of the random vector

$$v^\omega = (v_{1\omega}^1, \ldots, v_{h\omega}^h)$$

consists of $h$ independent component outcomes. The set

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_h$$

represents the crossing of sets $\Omega_i$. Assuming each set $\Omega_i$ contains $W_i$ possible outcomes, $|\Omega_i| = W_i$, the set $\Omega$ contains $W = \prod W_i$ elements, where $|\Omega| = W$ represents the number of all possible outcomes of the random vector $V$. Based on independence, the joint probability is the product

$$p^\omega = p_{1\omega}^1 p_{2\omega}^2 \cdots p_{h\omega}^h.$$
5.13.1.6 Using Monte Carlo Sampling

As noted above, for many practical problems it is impossible to obtain the universe solution, because the number of possible realizations $|\Omega|$ is way too large. The power of DECIS lies in its ability to compute excellent approximate solutions by employing Monte Carlo sampling techniques. Instead of computing the expected cost and the coefficients and the right-hand sides of the Benders cuts exactly (as it is done when solving the universe problem), DECIS, when using Monte Carlo sampling, estimates the quantities in each iteration using an independent sample drawn from the distribution of the random parameters. In addition to using crude Monte Carlo, DECIS uses importance sampling or control variates as variance reduction techniques.

The details of the algorithm and the different techniques used are described in the DECIS User's Manual. You can choose crude Monte Carlo, referred to as strategy 6, Monte Carlo importance sampling, referred to as strategy 2, or control variates, referred to as strategy 10. Both Monte Carlo importance sampling and control variates have been shown for many problems to give a better approximation compared to employing crude Monte Carlo sampling.

When using Monte Carlo sampling DECIS computes a close approximation to the true solution of the problem, and estimates a close approximation of the true optimal objective value. It also computes a confidence interval within which the true optimal objective of the problem lies, say with 95% confidence. The confidence interval is based on rigorous statistical theory. An outline of how the confidence interval is computed is given in the DECIS User's Manual. The size of the confidence interval depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for the estimation. You can expect the confidence interval to be very small, especially when you employ importance sampling or control variates as a variance reduction technique.

When employing Monte Carlo sampling techniques you have to choose a sample size (set in the parameter file). Clearly, the larger the sample size the better will be the approximate solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Setting the sample size too small may lead to bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

5.13.1.7 Monte Carlo Pre-sampling

We refer to pre-sampling when we first take a random sample from the distribution of the random parameters and then generate the approximate stochastic problem defined by the sample. The obtained approximate problem is then solved exactly using decomposition. This is in contrast to the way we used Monte Carlo sampling in the previous section, where we used Monte Carlo sampling in each iteration of the decomposition.

The details of the techniques used for pre-sampling are discussed in the DECIS User's Manual. DECIS computes the exact solution of the sampled problem using decomposition. This solution is an approximate solution of the original stochastic problem. Besides this approximate solution, DECIS computes an estimate of the expected cost corresponding to this approximate solution and a confidence interval within which the true optimal objective of the original stochastic problem lies with, say, 95% confidence. The confidence interval is based on statistical theory, while its size depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for generating the approximate problem. In conjunction with pre-sampling no variance reduction techniques are currently implemented.

Using Monte Carlo pre-sampling you have to choose a sample size. Clearly, the larger the sample size you choose, the better will be the solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Again, setting the sample size too small may lead to a bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

To use Monte Carlo pre-sampling choose strategy 8.
5.13.1.8 Regularized Decomposition

When solving practical problems, the number of Benders iterations can be quite large. In order to control the decomposition, with the hope to reduce the iteration count and the solution time, DECIS makes use of regularization. When employing regularization, an additional quadratic term is added to the objective of the master problem, representing the square of the distance between the best solution found so far (the incumbent solution) and the variable $x$. Using this term, DECIS controls the distance between solutions in successive decomposition iterations.

To enable regularization you have to set the corresponding parameter. You also have to choose the value of the constant rho in the regularization term. The default is regularization disabled. Details of how DECIS carries out regularization are represented in the DECIS User's Manual.

Regularization is only implemented when using MINOS as the optimizer for solving subproblems. Regularization has proven to be helpful for problems that need a large number of Benders iteration when solved without regularization. Problems that need only a small number of Benders iterations without regularization are not expected to improve much with regularization, and may need even more iterations with regularization than without.

5.13.2 GAMS/DECIS

GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS spares you from worrying about all the details of the input formats. It makes the problem formulation much easier but still gives you almost all the flexibility of using DECIS directly.

The link from GAMS to DECIS has been designed in such a way that almost no extensions to the GAMS modeling language were necessary for carrying out the formulation and solution of stochastic programs. In a future release of GAMS, however, additions to the language are planned that will allow you to model stochastic programs in an even more elegant way.

5.13.2.1 Setting up a Stochastic Program Using GAMS/DECIS

The interface from GAMS to DECIS supports the formulation and solution of stochastic linear programs. DECIS solves them using two-stage decomposition. The GAMS/DECIS interface resembles closely the structure of the SMPS (stochastic mathematical programming interface) discussed in the DECIS User's Manual. The specification of a stochastic problem using GAMS/DECIS uses the following components:

- the deterministic (core) model,
- the specification of the decision stages,
- the specification of the random parameters, and
- setting DECIS as the optimizer to be used.
5.13.2.2 Starting with the Deterministic Model

The core model is a deterministic linear program where all random parameters are replaced by their mean or by a particular realization. One could also see it as a GAMS model without any randomness. It could be a deterministic model that you have, that you intend to expand to a stochastic one. Using DECIS with GAMS allows you to easily extend a deterministic linear programming model to a stochastic one. For example, the following GAMS model represents a deterministic version of the electric power expansion planning illustrative example discussed in Infanger (1994).

* APL1P test model
* Dr. Gerd Infanger, November 1997
* Deterministic Program

set g generators / g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
   h   m   l
  g1 4.3 2.0 0.5
  g2 8.7 4.0 1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
   cost total cost
   cmin(g) minimum capacity
   cmax(g) maximum capacity
   omax(g) maximum operating level
   demand(dl) satisfy demand;

   cost .. tcost =e= sum(g, c(g)*x(g))
            + sum(g, sum(dl, f(g,dl)*y(g,dl)))
            + sum(dl,us(dl)*s(dl));

   cmin(g) .. x(g) =g= ccmin(g);
   cmax(g) .. x(g) =l= ccmax(g);
   omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
   demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

option lp=minos5;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;

5.13.2.3 Setting the Decision Stages

Next in order to extend a deterministic model to a stochastic one you must specify the decision stages. DECIS solves stochastic programs by two-stage decomposition. Accordingly, you must specify which variables belong to the first stage and which to the second stage, as well as which constraints are first-stage constraints and which are second-stage constraints. First stage constraints involve only first-stage variables, while second-stage constraints involve both first- and second-stage variables. You must specify the stage of a variable or a constraint by setting the stage suffix ".STAGE" to either one or two depending on if it is a first or second stage variable or constraint. For example, expanding the illustrative model above by

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
omax.stage(g) = 2;
demand.stage(dl) = 2;

would make x(g) first-stage variables, y(g, dl) and s(dl) second-stage variables, cmin(g) and cmax(g) first-stage constraints, and omax(g) and demand(g) second-stage constraints. The objective is treated separately, you don’t need to set the stage suffix for the objective variable and objective equation.

Note that the use of the .stage variable and equation suffix causes the GAMS scaling facility (i.e. the .scale suffix) to be unavailable. Stochastic models have to be scaled manually.

5.13.2.4 Specifying the Stochastic Model

DECIS supports any linear dependency model, i.e., the outcomes of an uncertain parameter in the linear program are a linear function of a number of independent random parameter outcomes. DECIS considers only discrete distributions: you must approximate any continuous distributions by discrete ones. The number of possible realizations of the discrete random parameters determines the accuracy of the approximation. A special case of a linear dependency model arises when you have only independent random parameters in your model. In this case the independent random parameters are mapped one to one into the random parameters of the stochastic program. We will present the independent case first and then expand to the case with linear dependency. According to setting up a linear dependency model we present the formulation in GAMS by first defining independent random parameters and then defining the distributions of the uncertain parameters in your model.

5.13.2.4.1 Specifying Independent Random Parameters

There are of course many different ways you can set up independent random parameters in GAMS. In the following we show one possible way that is generic and thus can be adapted for different models. The set-up uses the set stoch for labeling outcome named "out" and probability named "pro" of each independent random parameter. In the following we show how to define an independent random parameter, say, v1. The formulation uses the set omegal as driving set, where the set contains one element for each possible realization the random parameter can assume. For example, the set omegal has four elements according to a discrete distribution of four possible outcomes. The distribution of the random parameter is defined as the parameter v1, a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible
realizations of the set omega1, "o11", "o12", "o13", and "o14". For example, the random parameter v1 has outcomes of −1.0, −0.9, −0.5, −0.1 with probabilities 0.2, 0.3, 0.4, 0.1, respectively. Instead of using assignment statements for inputting the different realizations and corresponding probabilities you could also use the table statement. Always make sure that the sum of the probabilities of each independent random parameter adds to one.

* defining independent stochastic parameters

```gams
set stoch /out, pro /;
set omega1 / o11, o12, o13, o14 /;
table v1(stoch, omega1)
      o11 o12 o13 o14
out   -1.0 -0.9 -0.5 -0.1
pro    0.2  0.3  0.4  0.1
;
```

Random parameter v1 is the first out of five independent random parameters of the illustrative model APL1P, where the first two represent the independent availabilities of the generators g1 and g2 and the latter three represent the independent demands of the demand levels h, m, and l. We also represent the definitions of the remaining four independent random parameters. Note that random parameters v3, v4, and v5 are identically distributed.

```gams
set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21 o22 o23 o24 o25
out   -1.0 -0.9 -0.7 -0.1 -0.0
pro    0.1  0.2  0.5  0.1  0.1
;
```

```gams
set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11 o12 o13 o14
out   900  1000 1100 1200
pro    0.15 0.45 0.25 0.15
;
```

```gams
set omega4 / o41, o42, o43, o44 /;
table v4(stoch, omega1)
      o11 o12 o13 o14
out   900  1000 1100 1200
pro    0.15 0.45 0.25 0.15
;
```

```gams
set omega5 / o51, o52, o53, o54 /;
table v5(stoch, omega1)
      o11 o12 o13 o14
out   900  1000 1100 1200
pro    0.15 0.45 0.25 0.15
;
```

5.13.2.4.2 Defining the Distributions of the Uncertain Parameters in the Model Having defined the independent stochastic parameters (you may copy the setup above and adapt it for your model), we next define the stochastic parameters in the GAMS model. The stochastic parameters of the
model are defined by writing a file, the GAMS stochastic file, using the put facility of GAMS. The GAMS stochastic file resembles closely the stochastic file of the SMPS input format. The main difference is that we use the row, column, bounds, and right hand side names of the GAMS model and that we can write it in free format.

Independent Stochastic Parameters

First we describe the case where all stochastic parameters in the model are independent, see below the representation of the stochastic parameters for the illustrative example APL1P, which has five independent stochastic parameters.

First define the GAMS stochastic file "MODEL.STG" (only the exact name in uppercase letters is supported) and set up GAMS to write to it. This is done by the first two statements. You may want to consult the GAMS manual for how to use put for writing files. The next statement "INDEP DISCRETE" indicates that a section of independent stochastic parameters follows. We then write all possible outcomes and corresponding probabilities for each stochastic parameter by using loop statements. Of course one could also write each line separately, but the loops work nicely. Writing a "*" between the definitions of the independent stochastic parameters is merely for optical reasons and can be omitted.

* defining distributions (writing file MODEL.STG)
file stg /MODEL.STG/;
put stg;

put "INDEP DISCRETE" /;
loop(omega1,
  put "x g1 omax g1 ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
  put "*" /;
  loop(omega2,
    put "x g2 omax g2 ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
    put "*" /;
    loop(omega3,
      put "RHS demand h ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
      put "*" /;
      loop(omega4,
        put "RHS demand m ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
      )
      put "*" /;
      loop(omega5,
        put "RHS demand l ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
      )
    )
  )
)
putclose stg;

In the example APL1P the first stochastic parameter is the availability of generator g1. In the model the parameter appears as the coefficient of variable x(g1) in equation omax(g1). The definition using the put statement first gives the stochastic parameter as the intersection of variable x(g1) with equation omax(g1), but without having to type the braces, thus \( x g1 omax g1 \), then the outcome \( v1("out", omega1) \) and the probability \( v1("pro", omega1) \) separated by "period2". The different elements of the statement must be separated by blanks. Since the outcomes and probabilities of the first stochastic parameters are driven by the set omega1 we loop over all elements of the set omega1. We continue and define all possible outcomes for each of the five independent stochastic parameters.

In the example of independent stochastic parameters, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:
INDEP DISCRETE
x g1 omax g1 -1.00 period2 0.20
x g1 omax g1 -0.90 period2 0.30
x g1 omax g1 -0.50 period2 0.40
x g1 omax g1 -0.10 period2 0.10
*
x g2 omax g2 -1.00 period2 0.10
x g2 omax g2 -0.90 period2 0.20
x g2 omax g2 -0.70 period2 0.50
x g2 omax g2 -0.10 period2 0.10
x g2 omax g2 0.00 period2 0.10
*
RHS demand h 900.00 period2 0.15
RHS demand h 1000.00 period2 0.45
RHS demand h 1100.00 period2 0.25
RHS demand h 1200.00 period2 0.15
*
RHS demand m 900.00 period2 0.15
RHS demand m 1000.00 period2 0.45
RHS demand m 1100.00 period2 0.25
RHS demand m 1200.00 period2 0.15
*
RHS demand l 900.00 period2 0.15
RHS demand l 1000.00 period2 0.45
RHS demand l 1100.00 period2 0.25
RHS demand l 1200.00 period2 0.15

For defining stochastic parameters in the right-hand side of the model use the keyword RHS as the column name, and the equation name of the equation which right-hand side is uncertain, see for example the specification of the uncertain demands RHS demand h, RHS demand m, and RHS demand l. For defining uncertain bound parameters you would use the keywords UP, LO, or FX, the string bnd, and the variable name of the variable whose upper, lower, or fixed bound is uncertain.

Note all the keywords for the definitions are in capital letters, i.e., "INDEP DISCRETE", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that in GAMS equations, variables may appear in the right-hand side, e.g. "EQ. X+1 =L= 2*Y". When the coefficient 2 is a random variable, we need to be aware that GAMS will generate the following LP row X - 2*Y =L= -1. Suppose the probability distribution of this random variable is given by:

```gams
set s scenario /pessimistic, average, optimistic/;
parameter outcome(s) / pessimistic 1.5
  average 2.0
  optimistic 2.3 /;
parameter prob(s) / pessimistic 0.2
  average 0.6
  optimistic 0.2 /;
```

then the correct way of generating the entries in the stochastic file would be:

```gams
loop(s,
  put "Y EQ ",(-outcome(s))," PERIOD2 ",prob(s)/;
);
```

Note the negation of the outcome parameter. Also note that expressions in a PUT statement have to be surrounded by parentheses. GAMS reports in the row listing section of the listing file how equations are generated. You are encouraged to inspect the row listing to discover or confirm how coefficients appear in a generated LP row.
5.13.2.4.3 Dependent Stochastic Parameters  Next we describe the case of general linear dependency of the stochastic parameters in the model, see below the representation of the stochastic parameters for the illustrative example APL1PCA, which has three dependent stochastic demands driven by two independent stochastic random parameters. First we give the definition of the two independent stochastic parameters, which in the example happen to have two outcomes each.

* defining independent stochastic parameters
set stoch /out, pro/;
set omega1 / o11, o12 /;
table v1(stoch,omega1)
  o11  o12
  out  2.1  1.0
  pro  0.5  0.5 ;
set omega2 / o21, o22 /;
table v2(stoch, omega2)
  o21  o22
  out  2.0  1.0
  pro  0.2  0.8 ;

We next define the parameters of the transition matrix from the independent stochastic parameters to the dependent stochastic parameters of the model. We do this by defining two parameter vectors, where the vector \( hm1 \) gives the coefficients of the independent random parameter \( v1 \) in each of the three demand levels and the vector \( hm2 \) gives the coefficients of the independent random parameter \( v2 \) in each of the three demand levels.

parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /;

Again first define the GAMS stochastic file "MODEL.STG" and set GAMS to write to it. The statement BLOCKS DISCRETE indicates that a section of linear dependent stochastic parameters follows.

* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
  put "BL v1 period2 ", v1("pro", omega1)/;
  loop(dl,
    h1 = hm1(dl) * v1("out", omega1);
    put "RHS demand ", dl.tl:1, " ", h1/;
  );
);
loop(omega2,
  put " BL v2 period2 ", v2("pro", omega2) /;
  loop(dl,
    h1 = hm2(dl) * v2("out", omega2);
    put "RHS demand ", dl.tl:1, " ", h1/;
  );
);
putclose stg;
Dependent stochastic parameters are defined as functions of independent random parameters. The keyword BL labels a possible realization of an independent random parameter. The name besides the BL keyword is used to distinguish between different outcomes of the same independent random parameter or a different one. While you could use any unique names for the independent random parameters, it appears natural to use the names you have already defined above, e.g., v1 and v2. For each realization of each independent random parameter define the outcome of every dependent random parameter (as a function of the independent one). If a dependent random parameter in the GAMS model depends on two or more different independent random parameters the contributions of each of the independent parameters are added. We are therefore in the position to model any linear dependency model. (Note that the class of models that can be accommodated here is more general than linear. The functions, with which an independent random variable contributes to the dependent random variables can be any ones in one argument. As a general rule, any stochastic model that can be estimated by linear regression is supported by GAMS/DECIS.)

Define each independent random parameter outcome and the probability associated with it. For example, the statement starting with BL v1 period2 indicates that an outcome of (independent random parameter) v1 is being defined. The name period2 indicates that it is a second-stage random parameter, and v1("pro", omega1) gives the probability associated with this outcome. Next list all random parameters dependent on the independent random parameter outcome just defined. Define the dependent stochastic parameter coefficients by the GAMS variable name and equation name, or "RHS" and variable name, together with the value of the parameter associated with this realization. In the example, we have three dependent demands. Using the scalar h1 for internally storing the results of the calculation, looping over the different demand levels dl we calculate h1 = h1(dl) * v1("out", omega1) and define the dependent random parameters as the right-hand sides of equation demand(dl).

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when BL v1 appears the second time), a different outcome of the same independent random parameter is being defined, while a different block name (e.g., when BL v2 appears the first time) indicates that the first outcome of a different independent random parameter is being defined. You must ensure that the probabilities of the different outcomes of each of the independent random parameters add up to one. The loop over all elements of omega1 defines all realizations of the independent random parameter v1 and the loop over all elements of omega2 defines all realizations of the independent random parameter v2.

Note that for the first realization of an independent random parameter, you must define all dependent parameters and their realizations. The values entered serve as a base case. For any other realization of an independent random parameter you only need to define the dependent parameters that have different coefficients than have been defined in the base case. For those not defined in a particular realization, their values of the base case are automatically added.

In the example of dependent stochastic parameters above, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
BLOCKS DISCRETE
BL v1 period2 0.50
RHS demand h 630.00
RHS demand m 840.00
RHS demand l 420.00
BL v1 period2 0.50
RHS demand h 300.00
RHS demand m 400.00
RHS demand l 200.00
BL v1 period2 0.50
RHS demand h 200.00
RHS demand m 300.00
RHS demand l 600.00
BL v2 period2 0.20
RHS demand h 300.00
RHS demand m 400.00
RHS demand l 200.00
BL v2 period2 0.20
RHS demand h 200.00
RHS demand m 300.00
RHS demand l 600.00
BL v2 period2 0.80
RHS demand h 100.00
RHS demand m 150.00
RHS demand l 300.00
```
Again all the keywords for the definitions are in capital letters, i.e., "BLOCKS DISCRETE", "BL", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that you can only define random parameter coefficients that are nonzero in your GAMS model. When setting up the deterministic core model put a nonzero entry as a placeholder for any coefficient that you wish to specify as a stochastic parameter. Specifying a random parameter at the location of a zero coefficient in the GAMS model causes DECIS to terminate with an error message.

5.13.2.5 Setting DECIS as the Optimizer

After having finished the stochastic definitions you must set DECIS as the optimizer. This is done by issuing the following statements:

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX

```
option lp=decism;
aplp.optfile = 1;
```

The statement `option lp = decism` sets DECIS with the MINOS LP engine as the optimizer to be used for solving the stochastic problem. Note that if you do not use DECIS, but instead use any other linear programming optimizer, your GAMS model will still run and optimize the deterministic core model that you have specified. The statement `aplp.optfile = 1` directs GAMS to process the file `DECIS.OPT`, in which you may define any DECIS parameters.

5.13.2.5.1 Setting Parameter Options in the GAMS Model  The options iteration limit and resource limit can be set directly in your GAMS model file. For example, the following statements

```
option iterlim = 1000;
option reslim = 6000;
```

limit the number of decomposition iterations to be less than or equal to 1000, and the elapsed time for running DECIS to be less than or equal to 6000 seconds or 100 minutes.

5.13.2.5.2 Setting Parameters in the DECIS Options File  In the DECIS options file you can specify parameters regarding the solution algorithm used and control the output of the DECIS program. There is a record for each parameter you want to specify. Each record consists of the value of the parameter you want to specify and the keyword identifying the parameter, separated by a blank character or a comma. You may specify parameters with the following keywords: istrat, nsamples, nzrows, iwrite, ibug, iscratch, ireg, rho, tolben, and tolw in any order. Each keyword can be specified in lower case or upper case text in the format (A10). Since DECIS reads the records in free format you don't have to worry about the format, but some computers require that the text is inputted in quotes. Parameters that are not specified in the parameter file automatically assume their default values. See details of these parameters from Section Description of GAMS/DECIS Options.

5.13.2.5.3 Example  In the following example the parameters istrat = 7, nsamples = 200, and nzrows = 200 are specified. All other parameters are set at their default values. DECIS first solves the expected value problem and then the stochastic problem using crude Monte Carlo sampling with a sample size of nsamples = 200. DECIS reserves space for a maximum of nzrows = 50 cuts.

```
7 "ISTRAT"
200 "NSAMPLES"
50 "NZROWS"
```
5.13.2.5.4 Setting MINOS Parameters in the MINOS Specification File  When you use MINOS as the optimizer for solving the master and the subproblems, you must specify optimization parameters in the MINOS specification file "MINOS.SPC". Each record of the file corresponds to the specification of one parameter and consists of a keyword and the value of the parameter in free format. Records having a "∗" as their first character are considered as comment lines and are not further processed. For a detailed description of these parameters, see the MINOS Users' Guide (Murtagh and Saunders (1983) [142]). The following parameters should be specified with some consideration:

- **AIJ TOLERANCE** — Specifies the nonzero tolerance for constraint matrix elements of the problem. Matrix elements $a_{ij}$ that have a value for which $|a_{ij}|$ is less than "AIJ TOLERANCE" are considered by MINOS as zero and are automatically eliminated from the problem. It is wise to specify "AIJ TOLERANCE 0.0"

- **SCALE** — Specifies MINOS to scale the problem ("SCALE YES") or not ("SCALE NO"). It is wise to specify "SCALE NO".

- **ROWS** — Specifies the number of rows in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ROWS" should be specified as the number of constraints in the core problem or greater.

- **COLUMNS** — Specifies the number of columns in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "COLUMNS" should be specified as the number of variables in the core problem or greater.

- **ELEMENTS** — Specifies the number of nonzero matrix coefficients in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ELEMENTS" should be specified as the number of nonzero matrix coefficients in the core problem or greater.

5.13.2.5.5 Example  The following example represents typical specifications for running DECIS with MINOS as the optimizer.

```
BEGIN SPECS
PRINT LEVEL 1
LOG FREQUENCY 10
SUMMARY FREQUENCY 10
MPS FILE 12
ROWS 20000
COLUMNS 50000
ELEMENTS 100000
ITERATIONS LIMIT 30000
*
FACTORIZATION FREQUENCY 100
AIJ TOLERANCE 0.0
*
SCALE NO
END OF SPECS
```

5.13.2.5.6 Setting CPLEX Parameters Using System Environment Variables  When you use CPLEX as the optimizer for solving the master and the subproblems, optimization parameters must be specified through system environment variables. You can specify the parameters "CPLEXLICDIR", "SCALELP", "NOPRESOLVE", "ITERLOG", "OPTIMALITYTOL", "FEASIBILITYTOL", and "DUALSIMPLEX".

- **CPLEXLICDIR** — Contains the path to the CPLEX license directory. For example, on a Unix system with the CPLEX license directory in /usr/users/cplex/cplexlicdir you issue the command `setenv CPLEXLICDIR /usr/users/cplex/cplexlicdir`. 

• SCALELP — Specifies CPLEX to scale the master and subproblems before solving them. If the environment variable is not set no scaling is used. Setting the environment variable, e.g., by issuing the command `setenv SCALELP yes`, scaling is switched on.

• NOPRESOLVE — Allows to switch off CPLEX's presolver. If the environment variable is not set, presolve will be used. Setting the environment variable, e.g., by setting `setenv NOPRESOLVE yes`, no presolve will be used.

• ITERLOG — Specifies the iteration log of the CPLEX iterations to be printed to the file "MODEL.CPX". If you do not set the environment variable no iteration log will be printed. Setting the environment variable, e.g., by setting `setenv ITERLOG yes`, the CPLEX iteration log is printed.

• OPTIMALITYTOL — Specifies the optimality tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting `setenv OPTIMALITYTOL 1.0E-7` sets the CPLEX optimality tolerance to 0.0000001.

• FEASIBILITYTOL — Specifies the feasibility tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting `setenv FEASIBILITYTOL 1.0E-7` sets the CPLEX optimality tolerance to 0.0000001.

• DUALSIMPLEX — Specifies the dual simplex algorithm of CPLEX to be used. If the environment variable is not set the primal simplex algorithm will be used. This is the default and works beautifully for most problems. If the environment variable is set, e.g., by setting `setenv DUALSIMPLEX yes`, CPLEX uses the dual simplex algorithm for solving both master and subproblems.

5.13.2.6 GAMS/DECIS Output

After successfully having solved a problem, DECIS returns the objective, the optimal primal and optimal dual solution, the status of variables (if basic or not), and the status of equations (if binding or not) to GAMS. In the case of first-stage variables and equations you have all information available in GAMS, as you would with any other solver, but instead of obtaining the optimal values for a deterministic core problem you actually have the optimal values for the stochastic problem. However, for second-stage variables and constraints the expected values of the optimal primal and optimal dual solution are reported. This saves space and is useful for the calculation of risk measures. However, the information as to what the optimal primal and dual solutions were in the different scenarios of the stochastic programs is not reported back to GAMS. In a future release of the GAMS/DECIS interface the GAMS language is planned to be extended to handle the scenario second-stage optimal primal and dual values at least for selected variables and equations.

While running, DECIS outputs important information about the progress of the execution to your computer screen. After successfully solving a problem, DECIS also outputs its optimal solution to the solution output file "MODEL.SOL". The debug output file "MODEL.SCR" contains important information about the optimization run, and the optimzer output files "MODEL.MO" (when using DECIS with MINOS) or "MODEL.CPX" (when using DECIS with CPLEX) contain solution output from the optimizer used. In the DECIS User's Guide you find a detailed discussion of how to interpret the screen output, the solution report and the information in the output files.

5.13.2.6.1 The Screen Output The output to the screen allows you to observe the progress of a DECIS run. After the program logo and the copyright statement, you see four columns of output being written to the screen as long as the program proceeds. The first column (from left to right) represents the iteration count, the second column the lower bound (the optimal objective of the master problem), the third column the best upper bound (exact value or estimate of the total expected cost of the best solution found so far), and the fourth column the current upper bound (exact value or estimate of the total expected cost of current solution). After successful completion, DECIS quits with "Normal Exit", otherwise, if an error has been encountered, the programs stops with the message "Error Exit".
5.13.2.6.2 Example  When solving the illustrative example APL1P using strategy 5, we obtain the following report on the screen:

```
THE DECIS SYSTEM
Copyright (c) 1989 -- 1999 by Dr. Gerd Infanger
All rights reserved.
iter lower  best upper  current upper
 0  -0.9935E+06
 1  -0.4626E+06  0.2590E+05  0.2590E+05
 2   0.2111E+05  0.2590E+05  0.5487E+06
 3   0.2170E+05  0.2590E+05  0.2697E+05
 4   0.2368E+05  0.2384E+05  0.2384E+05
 5   0.2370E+05  0.2370E+05  0.2697E+05
 6   0.2370E+05  0.2390E+05  0.2370E+05
 7   0.2403E+05  0.2470E+05  0.2470E+05
 8   0.2433E+05  0.2470E+05  0.2699E+05
 9   0.2441E+05  0.2470E+05  0.2499E+05
10   0.2453E+05  0.2470E+05  0.2499E+05
11   0.2455E+05  0.2470E+05  0.2499E+05
12   0.2461E+05  0.2467E+05  0.2467E+05
13   0.2461E+05  0.2467E+05  0.2467E+05
14   0.2461E+05  0.2465E+05  0.2465E+05
15   0.2463E+05  0.2465E+05  0.2465E+05
16   0.2463E+05  0.2465E+05  0.2465E+05
17   0.2464E+05  0.2465E+05  0.2465E+05
18   0.2464E+05  0.2464E+05  0.2464E+05
19   0.2464E+05  0.2464E+05  0.2464E+05
20   0.2464E+05  0.2464E+05  0.2464E+05
21   0.2464E+05  0.2464E+05  0.2464E+05
22   0.2464E+05  0.2464E+05  0.2464E+05
```

Normal Exit

5.13.2.6.3 The Solution Output File  The solution output file contains the solution report from the DECIS run. Its name is "MODEL.SOL". The file contains the best objective function value found, the corresponding values of the first-stage variables, the corresponding optimal second-stage cost, and a lower and an upper bound on the optimal objective of the problem. In addition, the number of universe scenarios and the settings for the stopping tolerance are reported. In the case of using a deterministic strategy for solving the problem, exact values are reported. When using Monte Carlo sampling, estimated values, their variances, and the sample size used for the estimation are reported. Instead of exact upper and lower bounds, probabilistic upper and lower bounds, and a 95% confidence interval, within which the true optimal solution lies with 95% confidence, are reported. A detailed description of the solution output file can be found in the DECIS User's Guide.

5.13.2.6.4 The Debug Output File  The debug output file contains the standard output of a run of DECIS containing important information about the problem, its parameters, and its solution. It also contains any error messages that may occur during a run of DECIS. In the case that DECIS does not complete a run successfully, the cause of the trouble can usually be located using the information in the debug output file. If the standard output does not give enough information you can set the debug parameter ibug in the parameter input file to a higher value and obtain additional debug output. A detailed description of the debug output file can be found in the DECIS User's Guide.

5.13.2.6.5 The Optimizer Output Files  The optimizer output file "MODEL.MO" contains all the output from MINOS when called as a subroutine by DECIS. You can specify what degree of detail should be outputted by setting the appropriate "PRINT LEVEL" in the MINOS specification file. The optimizer output file "MODEL.CPX" reports messages and the iteration log (if switchwd on using the environment variable) from CPLEX when solving master and sub problems.
5.13.3 Description of GAMS/DECIS Options

5.13.3.1 DECIS Solver Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBug</td>
<td>Debug output</td>
<td>0</td>
</tr>
<tr>
<td>IReg</td>
<td>Indicator for regularized decomposition - MINOS only</td>
<td>0</td>
</tr>
<tr>
<td>IScratch</td>
<td>Internal unit number for output and debug</td>
<td>17</td>
</tr>
<tr>
<td>IStrat</td>
<td>Defines the solution strategy used</td>
<td>3</td>
</tr>
<tr>
<td>IWrite</td>
<td>Subproblem output</td>
<td>0</td>
</tr>
<tr>
<td>NSamples</td>
<td>Sample size used for the estimation</td>
<td>100</td>
</tr>
<tr>
<td>NZRows</td>
<td>Number of rows reserved for cuts in the master problem</td>
<td>100</td>
</tr>
<tr>
<td>Rho</td>
<td>rho parameter of regularization term in the objective function</td>
<td>1000</td>
</tr>
<tr>
<td>TolBen</td>
<td>Tolerance for stopping the decomposition algorithm</td>
<td>1e-7</td>
</tr>
<tr>
<td>TolW</td>
<td>tolerance when writing debug solution output</td>
<td>1e-9</td>
</tr>
</tbody>
</table>

5.13.3.2 DECIS EMP SP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVSubSolver</td>
<td>Subsolver to run on expected value problem</td>
<td>auto</td>
</tr>
<tr>
<td>EVSubSolverOpt</td>
<td>Optfile value to pass to the subsolver for expected value problem</td>
<td>0</td>
</tr>
<tr>
<td>FileName</td>
<td>Filename of generated scalar GAMS model</td>
<td>auto</td>
</tr>
<tr>
<td>ScenarioLog</td>
<td>Select log option for scenario solves</td>
<td>0</td>
</tr>
<tr>
<td>SolveEVProb</td>
<td>Solve and report the expected value solution</td>
<td>0</td>
</tr>
<tr>
<td>SolveScenarios</td>
<td>Maximum number of scenarios solved</td>
<td>100</td>
</tr>
<tr>
<td>SubSolvePar</td>
<td>User defined GAMS parameters for subsolve</td>
<td></td>
</tr>
<tr>
<td>SubSolver</td>
<td>Subsolver to run</td>
<td>decisc</td>
</tr>
<tr>
<td>Terminate</td>
<td>Generate the GAMS source code of the reformulated Model</td>
<td></td>
</tr>
</tbody>
</table>

**EVSubSolver** (string): Subsolver to run on expected value problem ←

Also controls solver used for individual scenarios.

Default: auto

**EVSubSolverOpt** (integer): Optfile value to pass to the subsolver for expected value problem ←

Range: \( \{0, \ldots, 999\} \)

Default: 0
**FileName** *(string)*: Filename of generated scalar GAMS model

  Default: `auto`

**IBug** *(integer)*: Debug output

  Default: `0`
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DECIS does not write any debug output</td>
</tr>
<tr>
<td>1</td>
<td>Solution of the master problem on each iteration</td>
</tr>
<tr>
<td>2</td>
<td>Value 1 plus scenario index and the optimal objective value for each subproblem solved</td>
</tr>
<tr>
<td>3</td>
<td>Value 2 plus information regarding importance sampling</td>
</tr>
<tr>
<td>4</td>
<td>Value 3 plus optimal dual variables of the cuts</td>
</tr>
<tr>
<td>5</td>
<td>Value 4 plus coefficients and the right-hand side of the cuts</td>
</tr>
<tr>
<td>6</td>
<td>Value 5 plus dump of the master problem and the subproblem in MPS format</td>
</tr>
</tbody>
</table>

**IReg** *(boolean)*: Indicator for regularized decomposition - MINOS only

Default: 0

**IScratch** *(integer)*: Internal unit number for output and debug

Range: \{1, ..., \infty\}

Default: 17

**I Strat** *(integer)*: Defines the solution strategy used

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Solves the expected value problem</td>
</tr>
<tr>
<td>2</td>
<td>Solves the stochastic problem using Monte Carlo importance sampling</td>
</tr>
<tr>
<td>3</td>
<td>Refers to istrat = 1 plus istrat = 2</td>
</tr>
<tr>
<td>4</td>
<td>Solves the stochastic universe problem</td>
</tr>
<tr>
<td>5</td>
<td>Refers to istrat = 1 plus istrat = 4</td>
</tr>
<tr>
<td>6</td>
<td>Solves the stochastic problem using crude Monte Carlo sampling</td>
</tr>
<tr>
<td>7</td>
<td>Refers to istrat = 1 plus istrat = 6</td>
</tr>
<tr>
<td>8</td>
<td>Solves the stochastic problem using Monte Carlo pre-sampling</td>
</tr>
<tr>
<td>9</td>
<td>Refers to istrat = 1 plus istrat = 8</td>
</tr>
<tr>
<td>10</td>
<td>Solves the stochastic problem using control variates</td>
</tr>
<tr>
<td>11</td>
<td>Refers to istrat = 1 plus istrat = 10</td>
</tr>
</tbody>
</table>

**IWrite** *(integer)*: Subproblem output

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No optimizer output is written</td>
</tr>
<tr>
<td>1</td>
<td>Optimizer output is written to the file</td>
</tr>
</tbody>
</table>

**NSamples** *(integer)*: Sample size used for the estimation

Range: \{30, ..., \infty\}

Default: 100
NZRows (integer): Number of rows reserved for cuts in the master problem

Range: \(\{1, ..., \infty\}\)

Default: 100

Rho (real): rho parameter of regularization term in the objective function

Default: 1000

ScenarioLog (integer): Select log option for scenario solves

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Quiet</td>
</tr>
<tr>
<td>1</td>
<td>Summary</td>
</tr>
<tr>
<td>2</td>
<td>Full log</td>
</tr>
</tbody>
</table>

SolveEVProb (no value): Solve and report the expected value solution

SolveScenarios (integer): Maximum number of scenarios solved

Default: 100

SubSolvePar (string): User defined GAMS parameters for subsolve

SubSolver (string): Subsolver to run

Default: decisc

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>decisc</td>
<td>Decis using Cplex</td>
</tr>
<tr>
<td>decism</td>
<td>Decis using Minos</td>
</tr>
</tbody>
</table>

Terminate (no value): Generate the GAMS source code of the reformulated Model

TolBen (real): Tolerance for stopping the decomposition algorithm

Default: 1e-7

TolW (real): tolerance when writing debug solution output

Default: 1e-9
5.13.4 Appendix A - GAMS/DECIS Illustrative Examples

5.13.4.1 Example APL1P

- APL1P test model
- Dr. Gerd Infanger, November 1997

```
set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
  cost total cost
  cmin(g) minimum capacity
  cmax(g) maximum capacity
  omax(g) maximum operating level
  demand(dl) satisfy demand;

  cost .. tcost =e= sum(g, c(g)*x(g))
         + sum(g, sum(dl, f(g,dl)*y(g,dl)))
         + sum(dl,us(dl)*s(dl));

  cmin(g) .. x(g) =g= ccmin(g);
  cmax(g) .. x(g) =l= ccmax(g);
  omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
  demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);
```

```
model apl1p /all/;

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
onmax.stage(g) = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro /;
set o11, o12, o13, o14 /;
table v1(stoch, o11)
  o11 o12 o13 o14
out -1.0 -0.9 -0.5 -0.1
pro 0.2 0.3 0.4 0.1

set o21, o22, o23, o24, o25 /;
table v2(stoch, o21)
  o21 o22 o23 o24 o25
out -1.0 -0.9 -0.7 -0.1 -0.0
pro 0.1 0.2 0.5 0.1 0.1
```
set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
   o11 o12 o13 o14
   out 900 1000 1100 1200
   pro 0.15 0.45 0.25 0.15
;
set omega4 / o41, o42, o43, o44 /;
table v4(stoch, omega1)
   o11 o12 o13 o14
   out 900 1000 1100 1200
   pro 0.15 0.45 0.25 0.15
;
set omega5 / o51, o52, o53, o54 /;
table v5(stoch, omega1)
   o11 o12 o13 o14
   out 900 1000 1100 1200
   pro 0.15 0.45 0.25 0.15
;
* defining distributions
file stg /MODEL.STG/;
pull stg;
pull "INDEP DISCRETE" /
loop(omega1,
pull "x g1 omax g1 ", v1("out", omega1), " period2 ", v1("pro", omega1) /
); pull "*" /
loop(omega2,
pull "x g2 omax g2 ", v2("out", omega2), " period2 ", v2("pro", omega2) /
); pull "*" /
loop(omega3,
pull "RHS demand h ", v3("out", omega3), " period2 ", v3("pro", omega3) /
); pull "*" /
loop(omega4,
pull "RHS demand m ", v4("out", omega4), " period2 ", v4("pro", omega4) /
); pull "*" /
loop(omega5,
pull "RHS demand l ", v5("out", omega5), " period2 ", v5("pro", omega5) /
); pullclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
aplmp.optfile = 1;
solve aplm using lp minimizing tcost;
scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
5.13.4.2 Example APL1PCA

* APL1PCA test model
* Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
   h  m  l
  g1  4.3  2.0  0.5
  g2  8.7  4.0  1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
cost total cost
cmin(g) minimum capacity
cmax(g) maximum capacity
omax(g) maximum operating level
demand(dl) satisfy demand;

cost .. tcost =e= sum(g, c(g)*x(g))
   + sum(g, sum(dl, f(g,dl)*y(g,dl)))
   + sum(dl,us(dl)*s(dl));

cmin(g) .. x(g) =g= ccmin(g);
cmax(g) .. x(g) =l= ccmax(g);
omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
omax.stage(g) = 1;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro/;
set omega1 / o11, o12 /;
table v1(stoch,omega1)
   o11 o12
out  2.1  1.0
   pro  0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
   o21 o22
out  2.0  1.0
   pro  0.2  0.8 ;

parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /;

* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /
scalar h1;
loop(omega1,
    put "BL v1 period2 ", v1("pro", omega1)/
    loop(dl,
        h1 = hm1(dl) * v1("out", omega1);
        put "RHS demand ", dl.tl:1, " ", h1;/
    );
);
    loop(omega2,
        put " BL v2 period2 ", v2("pro", omega2) /
        loop(dl,
            h1 = hm2(dl) * v2("out", omega2);
            put "RHS demand ", dl.tl:1, " ", h1;/
        );
    );
putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
5.13.5 Appendix B - Error Messages

1. ERROR in MODEL.STO: kwd, word1, word2 was not matched in first realization of block
   The specification of the stochastic parameters is incorrect. The stochastic parameter has not been
   specified in the specification of the first outcome of the block. When specifying the first outcome of
   a block always include all stochastic parameters corresponding to the block.

2. Option word1 word2 not supported
   You specified an input distribution in the stochastic file that is not supported. Check the DECIS
   manual for supported distributions.

3. Error in time file
   The time file is not correct. Check the file MODEL.TIM. Check the DECIS manual for the form of
   the time file.

4. ERROR in MODEL.STO: stochastic RHS for objective, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand
   side for the objective row (row name2). Check file MODEL.STO.

5. ERROR in MODEL.STO: stochastic RHS in master, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand
   side for the master problem (row name2). Check file MODEL.STO.

6. ERROR in MODEL.STO: col not found, name1
   The specification in the stochastic file is incorrect. The entry in the stochastic file, name1, is not
   found in the core file. Check file MODEL.STO.

7. ERROR in MODEL.STO: invalid col/row combination, (name1/name2)
   The stochastic file (MODEL.STO) contains an incorrect specification.

8. ERROR in MODEL.STO: no nonzero found (in B or D matrix) for col/row (name1, name2)
   There is no nonzero entry for the combination of name1 (col) and name2(row) in the B-matrix or in
   the D-matrix. Check the corresponding entry in the stochastic file (MODEL.STO). You may want
   to include a nonzero coefficient for (col/row) in the core file (MODEL.COR).

9. ERROR in MODEL.STO: col not found, name2
   The column name you specified in the stochastic file (MODEL.STO) does not exist in the core file
   (MODEL.COR). Check the file MODEL.STO.

10. ERROR in MODEL.STO: stochastic bound in master, col name2
    You specified a stochastic bound on first-stage variable name2. Check file MODEL.STO.

11. ERROR in MODEL.STO: invalid bound type (kwd) for col name2
    The bound type, kwd, you specified is invalid. Check file MODEL.STO.

12. ERROR in MODEL.STO: row not found, name2
    The specification in the stochastic file is incorrect. The row name, name2, does not exist in the core
    file. Check file MODEL.STO.

13. ERROR: problem infeasible
    The problem solved (master- or subproblem) turned out to be infeasible. If a subproblem is infeasible,
    you did not specify the problem as having the property of "complete recourse". Complete recourse
    means that whatever first-stage decision is passed to a subproblem, the subproblem will have a
    feasible solution. It is the best way to specify a problem, especially if you use a sampling based
    solution strategy. If DECIS encounters a feasible subproblem, it adds a feasibility cut and continues
    the execution. If DECIS encounters an infeasible master problem, the problem you specified is
    infeasible, and DECIS terminates. Check the problem formulation.

14. ERROR: problem unbounded
    The problem solved (master- or subproblem) turned out to be unbounded. Check the problem
    formulation.
15. ERROR: error code: inform
The solver returned with an error code from solving the problem (master- or subproblem). Consult the users’ manual of the solver (MINOS or CPLEX) for the meaning of the error code, inform. Check the problem formulation.

16. ERROR: while reading SPECS file
The MINOS specification file (MINOS.SPC) contains an error. Check the specification file. Consult the MINOS user’s manual.

17. ERROR: reading mps file, mpsfile
The core file mpsfile (i.e., MODEL.COR) is incorrect. Consult the DECIS manual for instructions regarding the MPS format.

18. ERROR: row 1 of problem is not a free row
The first row of the problem is not a free row (i.e., is not the objective row). In order to make the first row a free row, set the row type to be 'N'. Consult the DECIS manual for the MPS specification of the problem.

19. ERROR: name not found = nam1, nam2
There is an error in the core file (MODEL.COR). The problem cannot be decomposed correctly. Check the core file and check the model formulation.

20. ERROR: matrix not in staircase form
The constraint matrix of the problem as specified in core file (MODEL.COR) is not in staircase form. The first-stage rows and columns and the second-stage rows and columns are mixed within each other. Check the DECIS manual as to how to specify the core file. Check the core file and change the order of rows and columns.
5.13.6 DECIS License and Warranty

The software, which accompanies this license (the "Software") is the property of Gerd Infanger and is protected by copyright law. While Gerd Infanger continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum, which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may
  1. Use one copy of the Software on a single computer,
  2. Make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes,
  3. Use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network. After a written notice to Gerd Infanger, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement.

- You may not
  1. Copy the documentation, which accompanies the Software,
  2. Sublicense, rent or lease any portion of the Software,
  3. Reverse engineer, de-compile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software.

5.13.6.1 Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will be free from defects for a period of thirty (30) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of the warranty will be that Gerd Infanger will, at his option, replace any defective media returned to Gerd Infanger within the warranty period or refund the money you paid for the Software. Gerd Infanger does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

5.13.6.2 Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL GERD INFANGER BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF GERD INFANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO CASE SHALL GERD INFANGER'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.
5.13.6.3 General:

This Agreement will be governed by the laws of the State of California. This Agreement may only be modified by a license addendum, which accompanies this license or by a written document, which has been signed by both you and Gerd Infanger. Should you have any questions concerning this Agreement, or if you desire to contact Gerd Infanger for any reason, please write:

Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002, USA.

Note

Copyright © 1989 – 1999 by Gerd Infanger. All rights reserved. The GAMS/DECIS User’s Guide is copyrighted and all rights are reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Gerd Infanger. The DECIS software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement. The DECIS software can be licensed through Infanger Investment Technology, LLC or through Gams Development Corporation.

5.14 DICOPT

Author

Ignacio E. Grossmann, Jagadisan Viswanathan, Aldo Vecchietti; Engineering Research Design Center, Carnegie Mellon University, Pittsburgh, PA
Ramesh Raman, Erwin Kalvelagen; GAMS Development Corporation, Washington D.C.

5.14.1 Introduction

DICOPT is a program for solving mixed-integer nonlinear programming (MINLP) problems that involve linear binary or integer variables and linear and nonlinear continuous variables. While the modeling and solution of MINLP optimization problems has not yet reached the stage of maturity and reliability achieved by linear, integer, or non-linear programming modeling, these problems still have rich areas of application. For example, they often arise in engineering design, management sciences, and finance. DICOPT (DIscrete and Continuous OPTimizer) was developed by J. Viswanathan and Ignacio E. Grossmann at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside DICOPT solves a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under GAMS. The performance will heavily depend on the choice of the selected subsolvers.

Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum.

The GAMS/DICOPT system has been designed with two main goals in mind:

- to build on existing modeling concepts, introduce a minimum of extensions to the existing modeling language, and provide upward compatibility to ensure easy transition from existing modeling applications to nonlinear mixed-integer formulations
- to use existing optimizers to solve the DICOPT sub-problems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MIP solvers become automatically and immediate available to DICOPT.
5.14.2 Requirements

In order to use **DICOPT** you will need to have access to a licensed GAMS BASE system as well as at least one licensed MIP solver and one licensed NLP solver. For difficult models it is advisable to also have access to multiple solvers.

5.14.3 How to Run a Model with GAMS/DICOPT

**DICOPT** is only able to solve MINLP and MIQCP models. If you did not specify **DICOPT** as the default solver you can use the following statement in your GAMS model:

```
option minlp = dicopt;
```

This should appear before the solve statement. **DICOPT** automatically uses the default MIP and NLP solver to solve its sub-problems. One can override this with GAMS statements like:

```
option nlp = conopt;  { or any other nlp solver }
option mip = gurobi;  { or any other mip solver }
```

These options can also be specified on the command line:

```
> gams mymodel minlp=dicopt nlp=conopt mip=gurobi
```

In the Integrated Development Environment (IDE) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include **conopt**, **ipopt**, **knitro**, **minos**, and **snopt**. Possible MIP solvers include **cplex**, **gurobi**, and **xpress**.

With an option file it is even possible to use alternate solvers in different cycles. Section **DICOPT Options** explains this in detail.

5.14.4 Overview of DICOPT

**DICOPT** solves models of the form:

\[
\begin{align*}
\text{min or max} & \quad f(x, y) \\
\text{subject to} & \quad g(x, y) \sim b \\
& \quad \ell_x \leq x \leq u_x \\
& \quad y \in [\ell_y, \ldots, u_y]
\end{align*}
\]  

(MINLP)

where \(x\) are the continuous variables and \(y\) are the discrete variables. The symbol \(\sim\) is used to denote a vector of relational operators \(\leq, =, \geq\). The constraints can be either linear or non-linear. Bounds \(\ell\) and \(u\) on the variables are handled directly. \([x]\) indicates the smallest integer, greater than or equal to \(x\). Similarly, \([-x]\) indicates the largest integer, less than or equal to \(x\). The discrete variables can be either integer variables or binary variables.
5.14 DICOPT

5.14.5 The Algorithm

The algorithm in DICOPT is based on three key ideas:

- Outer Approximation
- Equality Relaxation
- Augmented Penalty

Outer Approximation refers to the fact that the surface described by a convex function lies above the tangent hyper-plane at any interior point of the surface. (In 1-dimension, the analogous geometrical result is that the tangent to a convex function at an interior point lies below the curve). In the algorithm outer-approximations are attained by generating linearizations at each iterations and accumulating them in order to provide successively improved linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region.

Equality Relaxation is based on the following result from non-linear programming. Suppose the MINLP problem is formulated in the form:

\[
\begin{align*}
\text{minimize or maximize} & \quad f(x) + c^T y \\
\text{subject to} & \quad G(x) + Hy \sim b \\
& \quad \ell \leq x \leq u \\
& \quad y \in \{0, 1\}
\end{align*}
\]

i.e. the discrete variables are binary variables and they appear linearly in the model.

Let \( y^{(0)} \) be any fixed binary vector and let \( x^{(0)} \) be the solution of the corresponding NLP subproblem:

\[
\begin{align*}
\text{minimize} & \quad c^T y^{(0)} + f(x) \\
\text{subject to} & \quad A y^{(0)} + h(x) = 0 \\
& \quad B y^{(0)} + g(x) \leq 0 \\
& \quad \ell \leq x \leq u
\end{align*}
\]

Further let

\[
T^{(0)} = \text{diag}(t_{i,i}) \\
t_{i,i} = \text{sign}(\lambda_i)
\]

where \( \lambda_i \) is the Lagrange multiplier of the \( i \)-th equality constraint.

If \( f \) is pseudo-convex, \( h \) is quasi-convex, and \( g \) is quasi-convex, then \( x^0 \) is also the solution of the following NLP:

\[
\begin{align*}
\text{minimize} & \quad c^T y^{(0)} + f(x) \\
\text{subject to} & \quad T^{(0)}(A y^{(0)} + h(x)) \leq 0 \\
& \quad B y^{(0)} + g(x) \leq 0 \\
& \quad \ell \leq x \leq u
\end{align*}
\]

In colloquial terms, under certain assumptions concerning the convexity of the nonlinear functions, an equality constraint can be relaxed to be an inequality constraint. This property is used in the MIP master problem to accumulate linear approximations.
Augmented Penalty refers to the introduction of (non-negative) slack variables on the right hand sides of the just described inequality constraints and the modification of the objective function when assumptions concerning convexity do not hold.

The algorithm underlying DICOPT starts by solving the NLP in which the 0-1 conditions on the binary variables are relaxed. If the solution to this problem yields an integer solution the search stops. Otherwise, it continues with an alternating sequence of nonlinear programs (NLP) called subproblems and mixed-integer linear programs (MIP) called master problems. The NLP subproblems are solved for fixed 0-1 variables that are predicted by the MIP master problem at each (major) iteration. For convex problems the master problem also provides a lower bound on the objective function. This lower bound (in the case of minimization) increases monotonically as iterations proceed due to the accumulation of linear approximations. Note that in the case of maximization this bound is an upper bound which can be used as a stopping criterion through a DICOPT option stop 1 (see section DICOPT Options). Another stopping criterion that tends to work very well for non-convex problems (and even for convex problems) is based on the heuristic: stop as soon as the NLP subproblems start worsening (i.e. the current NLP subproblem has an optimal objective function that is worse than the previous NLP subproblem). This stopping criterion relies on the use of the augmented penalty, and is used in the description of the algorithm below. This is also the default stopping criterion in the implementation of DICOPT. The algorithm can be stated briefly as follows:

1. Solve the NLP relaxation of the MINLP program. If \( y^{(0)} = y \) is integer, stop(integer optimum found). Else continue with step 2.
2. Find an integer point \( y^{(1)} \) with an MIP master problem that features an augmented penalty function to find the minimum over the convex hull determined by the half-spaces at the solution \((x^{(0)}, y^{(0)})\).
3. Fix the binary variables \( y = y^{(1)} \) and solve the resulting NLP. Let \((x^{(1)}, y^{(1)})\) be the corresponding solution.
4. Find an integer solution \( y^{(2)} \) with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at \( y^{(0)} \) and \( y^{(1)} \).
5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e. half-spaces - at the new KKT point).

In the MIP problems integer cuts are added to the model to exclude previously determined integer vectors \( y^{(1)}, y^{(2)}, ..., y^{(K)} \).

For a detailed description of the theory and references to earlier work, see [193] [110] [52].

The algorithm has been extended to handle general integer variables and integer variables appearing nonlinearly in the model.

### 5.14.6 Modeling

#### 5.14.6.1 Relaxed Model

Before solving a model with DICOPT, the user is strongly advised to experiment with the relaxed model where the integer restrictions are ignored. This is the RMINLP model. As the DICOPT will start solving the relaxed problem and can use an existing relaxed optimal solution, it is a good idea to solve the RMINLP always before attempting to solve the MINLP model, i.e., the following fragment is not detrimental with respect to performance:
The second \texttt{SOLVE} statement will only be executed if the first \texttt{SOLVE} was successful, i.e., if the model status was one (optimal) or two (locally optimal).

In general it is not a good idea to try to solve an MINLP model if the relaxed model cannot be solved reliably. As the RMINLP model is a normal NLP model, some obvious points of attention are:

- Scaling. If a model is poorly scaled an NLP solver may not be able find the optimal or even a feasible solution. Some NLP solvers have automatic scaling algorithms, but often it is better to attack this problem on the modeling level. The GAMS scaling facility can help in this respect.

- Starting point. If a poor starting point is used, the NLP solver may not be able to find a feasible or optimal solution. A starting point can be set by setting level values, e.g. \texttt{X.L = 1;} . The GAMS default levels are zero, which is often not a good choice.

- Adding bounds. Add bounds so that all functions can be properly evaluated. If you have a function $\sqrt{x}$ or $\log(x)$ in the model, you may want to add a bound \texttt{X.LO=0.001;} . If a function like $\log(f(x))$ is used, you may want to introduce an auxiliary variable and equation $y = f(x)$ with an appropriate bound \texttt{Y.LO=0.001;} .

In some cases the relaxed problem is the most difficult model. If more than one NLP solver is available you may want to try them in a sequence:
In this fragment we first try to solve the relaxed model using CONOPT. If that fails we try MINOS, and if that solve also fails we try SNOPT.

It is worthwhile to spend some time getting the relaxed model to solve reliably and speedily. In most cases, modeling improvements in the relaxed model, such as scaling, will also benefit the subsequent NLP sub-problems. In general these modeling improvements turn out to be rather solver independent: changes that improve the performance with CONOPT will also help solving the model with MINOS.

5.14.6.2 OPTCR and OPTCA

The DICOPT algorithm assumes that the integer sub-problems are solved to optimality. The GAMS options for OPTCR and OPTCA are therefore ignored: subproblems are solved with both tolerances set to zero. If you want to solve a MIP sub-problem with an optimality tolerance you can use the DICOPT option file to set OPTCR or OPTCA. For more information see section DICOPT Options.

For models with many discrete variables, it may be necessary to introduce an OPTCR or OPTCA option in order to solve the model in acceptable time. For models with a limited number of integer variables the default to solve MIP sub-models to optimality may be acceptable.

5.14.6.3 Integer Formulations

A number of MIP formulations are not very obvious and pose a demand on the modeler’s knowledge and experience. A good overview of integer programming modeling is given in [202].

Many integer formulations use a so-called big- $M$ construct. It is important to choose small values for those big- $M$ numbers. As an example consider the fixed charge problem where $y_i \in \{0,1\}$ indicate if facility $i$ is open or closed, and where $x_i$ is the production at facility $i$. Then the cost function can be modeled as:

$$
C_i = f_i y_i + v_i x_i
$$

$$
x_i \leq M_i y_i
$$

$$
y_i \in \{0,1\}
$$

$$
0 \leq x_i \leq \text{cap}_i
$$

where $f_i$ is the fixed cost and $v_i$ the variables cost of operating a facility $i$. In this case the chosen $M_i$ should be large enough that $x_i$ is not restricted if $y_i = 1$. On the other hand, it should be as small as possible. This leads to a choice to have $M_i$ equal to the (tight) upperbound of variable $x_i$ (i.e. the capacity $\text{cap}_i$ of facility $i$).

5.14.6.4 Non-smooth Functions

GAMS alerts NLP modelers against the use of non-smooth functions such as $\min()$, $\max()$, $\text{smin}()$, $\text{smax}()$ and $\text{abs}()$. In order to use these functions, a non-linear program needs to be declared as a DNLP model instead of a regular NLP model:

```gams
option dnlp=conopt;
model m /all/;
solve m minimizing z using dnlp;
```
This construct will warn the user that problems may arise due to the use of non-smooth functions.

A possible solution is to use a smooth approximation. For instance, the function \( f(x) = |x| \) can be approximated by \( g(x) = \sqrt{x^2 + \varepsilon} \) for some \( \varepsilon > 0 \). This approximation does not contain the point \((0,0)\). An alternative approximation can be devised that has this property:

\[
f(x) \approx \frac{2x}{1 + e^{-x/\delta}} - x
\]

MINLP models do not have such protection against non-smooth functions, but the use of such functions is just as problematic here. It is possible to use discrete variables with MINLP models in order to model if-then-else situations. For instance, in the case of the absolute value we can replace \( x \) by \( x^+ - x^- \) and \( |x| \) by \( x^+ + x^- \) by using:

\[
\begin{align*}
x &= x^+ - x^- \\
|x| &= x^+ + x^- \\
x^+ &\leq \delta M \\
x^- &\leq (1 - \delta) M \\
x^+, x^- &\geq 0 \\
\delta &\in \{0, 1\}
\end{align*}
\]

where \( \delta \) is a binary variable.

5.14.7 GAMS Options

GAMS options are specified in the GAMS model source, using either the option statement or a model suffix.

5.14.7.1 The OPTION Statement

An option statement sets a global parameter. An option statement should appear before the solve statement, as in:

```gams
model m /all/;
option iterlim=100;
solve m using minlp minimizing z;
```

Option statements that affect the behavior of DICOPT are listed below:

- **option domlim = n;**
  This option sets a limit on the total accumulated number of non-linear function evaluation errors that are allowed while solving the NLP subproblems or inside DICOPT itself. An example of a function evaluation error or domain error is taking the square root of a negative number. This situations can be prevented by adding proper bounds. The default is zero, i.e. no function evaluation errors are allowed.

In case a domain error occurs, the listing file will contain an appropriate message, including the equation that is causing the problem. For instance:

```
**** ERRORS(S) IN EQUATION loss(cc,sw)
    2 instance(s) of - UNDEFINED REAL POWER (RETURNED  0.0E+00)
```
If such errors appear you can increase the DOMLIM limit, but often it is better to prevent the errors from occurring in the first place. In many cases this can be accomplished by adding appropriate bounds. Sometimes extra variables and equations need to be added to accomplish this. For instance, with an expression like \( \log(x - y) \), you may want to introduce a variable \( z > \varepsilon \) and an equation \( z = x - y \), so that the expression can be rewritten as \( \log(z) \).

- **option iterlim = n;**
  This option sets a limit on the total accumulated (minor) iterations performed in the MIP and NLP subproblems.

- **option minlp = dicopt;**
  This option selects DICOPT to solve MINLP problems.

- **option mip = s;**
  This option sets the MIP solver to be used for the MIP master problems. Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

- **option nlp = s;**
  This option sets the NLP solver to be used for the NLP sub-problems. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

- **option optca = x;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

- **option optcr = x;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

- **option reslim = x;**
  This option sets a limit on the total accumulated time (in seconds) spent inside DICOPT and the subsolvers.

- **option sysout = on;**
  This option will print extra information to the listing file.

In the list above (and in the following) \( n \) indicates an integer number. GAMS will also accept fractional values, which will be rounded. Options marked with an \( x \) parameter expect a real number. Options with an \( s \) parameter expect a string argument.

### 5.14.7.2 The Model Suffix

Some options are set by assigning a value to a model suffix, as in:

```gams
model m /all/;
  m.optfile=1;
solve m using minlp minimizing z;
```

Model suffixes that affect the behavior of DICOPT are listed below:

- **m.dictfile = 1;**
  This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). Such information is needed when the DICOPT option nlptracelevel is used, otherwise the option can be ignored.
• `m.iterlim = n;`
  Sets the total accumulated (minor) iteration limit. This option overrides the global iteration limit
  set by an option statement, e.g.,

  ```
  model m /all/;
  m.iterlim = 100;
  option iterlim = 1000;
  solve m using minlp minimizing z;
  ```

  will cause DICOPT to use an iteration limit of 100.

• `m.optfile = 1;`
  This option instructs DICOPT to read an option file `dicopt.opt`. This file should be located in the
  current directory (or the project directory when using the GAMS IDE). The contents of the option
  file will be echoed to the listing file and to the screen (the log file):

  ```
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  > maxcycles 10
  --- DICOPT: Starting major iteration 1
  ```

  If the option file does not exist, the algorithm will proceed using its default settings. An appropriate
  message will be displayed in the listing file and in the log file:

  ```
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  --- DICOPT: File does not exist, using defaults...
  --- DICOPT: Starting major iteration 1
  ```

• `m.optfile = n;`
  If `n > 1` then the option file that is read is called `dicopt.op n` (for `n = 2,...,9`) or `dicopt.o n` (for
  `n = 10,...,99`). E.g. `m.optfile=2;` will cause DICOPT to read `dicopt.op2`.

• `m.prioropt = 1;`
  This option turns on the use of priorities on the discrete variables. Priorities influence the branching
  order chosen by the MIP solver during solution of the MIP master problems. The use of priorities
  can greatly impact MIP solver performance. The priorities themselves have to be specified using
  the `.prior` variables suffix, e.g. `x.prior(i,j) = ord(i);`. Contrary to intuition, variables with a
  lower value for their priority are branched on before variables with a higher priority, i.e., the most
  important variables should get lower priority values.

• `m.reslim = x;`
  Sets the total accumulated time limit. This option overrides the global time limit set by an option
  statement.

### 5.14.8 DICOPT Options

This sections describes the options that can be specified in the DICOPT option file. This file is usually
called `dicopt.opt`. The `optfile` model suffix must be set to tell DICOPT to read this file:

```
model m /all/;
    m.optfile=1;
    solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in the project directory when the IDE is used.

The option file is a standard text file, with a single option on each line. All options are case-insensitive. A
line is a comment line if it starts with an asterisk, `*` in column one. A valid option file can look like:
* stop only on infeasible MIP or hitting a limit
stop 0
* use minos to solve first NLP sub problem
* and conopt for all subsequent ones
nlpsolver minos conopt

A convenient way to write the option file from within a GAMS model is to use the following construct:

```
$onecho > dicopt.opt
stop 0
nlpsolver minos conopt
$offecho
```

This will make the model self-contained. Notice, however, that this overwrites an existing file `dicopt.opt`.

Available DICOPT options are listed below:

### 5.14.8.1 Algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>How to proceed in case of NLP errors</td>
<td>2</td>
</tr>
<tr>
<td>convex</td>
<td>If enabled, the defaults for a number of other options are set to values</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>appropriate to convex MINLPs</td>
<td></td>
</tr>
<tr>
<td>infbnd</td>
<td>Bound to use for unbounded integer variables in integer cuts</td>
<td>10000</td>
</tr>
<tr>
<td>infeasler</td>
<td>Use derivatives of infeasible nonlinear subproblems</td>
<td>0</td>
</tr>
<tr>
<td>maxcycles</td>
<td>Maximum number of cycles</td>
<td>20</td>
</tr>
<tr>
<td>relaxed</td>
<td>How to start DICOPT</td>
<td>1</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for NLP and MIP subsolver</td>
<td>5</td>
</tr>
<tr>
<td>stop</td>
<td>Stopping criterion</td>
<td>2</td>
</tr>
<tr>
<td>usexinit</td>
<td>Use the user initial point as starting point for all NLP solves</td>
<td>0</td>
</tr>
<tr>
<td>weight</td>
<td>Penalty parameter, set above 1e20 to disable penalty relaxation</td>
<td>1000</td>
</tr>
</tbody>
</table>

### 5.14.8.2 Tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsmip</td>
<td>Tolerance on test on monotonic improvement of MIP master problem</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>epex</td>
<td>Tolerance for integer values when loading relaxed solution</td>
<td>1.0e-3</td>
</tr>
</tbody>
</table>

### 5.14.8.3 MIP masterproblem options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mipiterlim</td>
<td>List of iteration limits</td>
<td></td>
</tr>
</tbody>
</table>
### 5.14 DICOPT

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mipoptfile</td>
<td>List of option files for MIP solver</td>
<td></td>
</tr>
<tr>
<td>mipreslim</td>
<td>List of resource limits</td>
<td></td>
</tr>
<tr>
<td>mipsolver</td>
<td>List of MIP solvers</td>
<td></td>
</tr>
<tr>
<td>optca</td>
<td>List of OPTCA values</td>
<td></td>
</tr>
<tr>
<td>optcr</td>
<td>List of OPTCR values</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.14.8.4 NLP subproblem options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>domlim</td>
<td>List of allowed number of domain errors</td>
<td></td>
</tr>
<tr>
<td>nlpiterlim</td>
<td>List of iteration limits</td>
<td></td>
</tr>
<tr>
<td>nlpoptfile</td>
<td>List of option files for NLP solver</td>
<td></td>
</tr>
<tr>
<td>nlpreslim</td>
<td>List of resource limits</td>
<td></td>
</tr>
<tr>
<td>nlpsolver</td>
<td>List of NLP solvers</td>
<td></td>
</tr>
<tr>
<td>nlptracefile</td>
<td>Base name of trace files</td>
<td>nlptrace</td>
</tr>
<tr>
<td>nlptracelevel</td>
<td>Trace level</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 5.14.8.5 Feasibility Pump

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>feasumpump</td>
<td>Whether to run the Feasibility Pump</td>
<td>0</td>
</tr>
<tr>
<td>fp_acttol</td>
<td>Tolerance on when to consider an equation as active</td>
<td>1E-6</td>
</tr>
<tr>
<td>fp_cutoffdecr</td>
<td>Additional relative decrement of cutoff value for the original objective function</td>
<td>0.1</td>
</tr>
<tr>
<td>fp_dumpsubprob</td>
<td>Whether to dump subproblems (NLPs and MIPs) into GAMS files</td>
<td>0</td>
</tr>
<tr>
<td>fp_integercuts</td>
<td>Whether to add integer cuts after an NLP subproblem has been solved to optimality</td>
<td>1</td>
</tr>
<tr>
<td>fp_iterlimit</td>
<td>Major Iteration limit</td>
<td>20</td>
</tr>
<tr>
<td>fp mipgap</td>
<td>Optimality tolerance (relative gap) to use for solving MIP projection problem</td>
<td>0.01</td>
</tr>
<tr>
<td>fp projcuts</td>
<td>Whether to add cut derived from projection of MIP solution onto NLP feasible set</td>
<td>1</td>
</tr>
<tr>
<td>fp projzerotol</td>
<td>Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP</td>
<td>1E-4</td>
</tr>
<tr>
<td>fp sollimit</td>
<td>Stop when a number of (improving) solutions has been bound</td>
<td>maxint</td>
</tr>
<tr>
<td>fp stalllimit</td>
<td>Stop when no improving solution has been bound for a number of FP iterations, -1 to disable</td>
<td>5</td>
</tr>
<tr>
<td>fp subsolverlog</td>
<td>Whether to show the log of subsolvers</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.14.8.6 Detailed Options Description

**continue (integer):** How to proceed in case of NLP errors

This option can be used to let DICOPT continue in cases of NLP solver failure. The preferred approach is to fix the model so that NLP subproblems solve without problems. In some cases, however, (partial) failures of an NLP solver in solving the NLP subproblems can be ignored, as DICOPT may recover later on. Adding the option `continue 0` during model debugging enables DICOPT to function in a more specific way.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
</table>
| 0     | Stop on solver failure  
Stop on solver failure. DICOPT will terminate when an NLP subproblem cannot be solved to optimality. Some NLP solvers terminate with a status other than optimal if not all of the termination criteria are met. For instance, the change in the objective function is negligible (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close to the (local) optimum. Using `continue 0` will prevent DICOPT from accepting such a solution. |
| 1     | Accept non-optimal feasible solutions  
NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver cannot make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option. |
| 2     | Ignore infeasible solutions  
NLP subproblem failures resulting in a non-optimal but feasible solution are accepted (as in option `continue 1`). NLP subproblem failures resulting in an infeasible solution are ignored. The corresponding configuration of discrete variables is forbidden to be used again. An integer cut to accomplish this is added to subsequent MIP master problems. Note that the relaxed NLP solution should be feasible. This setting is the default. |

**convex (boolean):** If enabled, the defaults for a number of other options are set to values appropriate to convex MINLPs

If this option is enabled, the default option values will be changed such that DICOPT will stop on crossover (`stop` set to 1), linearizations from infeasible NLP subproblems will be added to the MIP master problem (`infeasder` set to 1), penalty relaxation is no longer applied to linearizations (`weight` set to `maxdouble`), and the feasibility pump is run if there are no semicontinuous or semiinteger variables and no special ordered sets (`feaspump` set to 1).

Default: 0

**domlim (string):** List of allowed number of domain errors

---

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fp_timelimit</code></td>
<td>Time limit</td>
<td><code>maxdouble</code></td>
</tr>
<tr>
<td><code>fp_transfercuts</code></td>
<td>Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible)</td>
<td>1</td>
</tr>
</tbody>
</table>
domlim $i_1 \ldots i_n$. Sets a limit of the number of function and derivative evaluation errors for a particular cycle. A number of $-1$ means that the global GAMS option domlim is used. The last number $i_n$ sets a domain error limit for all cycles $n, n+1, \ldots$.

Example: domlim 0 100 0
The NLP solver in the second cycle is allowed to make up to 100 evaluation errors, while all other cycles must be solved without evaluation errors.

The default is to use the global GAMS domlim option.

**epsmip** *(real)*: Tolerance on test on monotonic improvement of MIP master problem

This option can be used to relax the test on MIP objective functions. The objective function values of the MIP master problems should form a monotonic worsening curve. This is not the case if the MIP master problems are not solved to optimality. If the options OPTCR or OPTCA are set to a nonzero value, this test is bypassed. If the test fails, DICOPT will fail with a message:

The MIP solution became better after adding integer cuts. Something is wrong. Please check if your model is properly scaled. Also check your big M formulations -- the value of M should be relatively small. This error can also occur if you used a MIP solver option file with a nonzero OPTCR or OPTCA setting. In that case you may want to increase the EPSMIP setting using a DICOPT option file.

The value of

$$\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|}$$

is compared against epsmip. In case the test fails but you want DICOPT to continue anyway, you may want to increase the value of epsmip. The current values used in the test (previous and current MIP objective, epsmip) are printed along with the above message to provide information about how much you should increase epsmip to pass the test. Normally, you should not have to change this value.

Default: $1.0e^{-6}$

**epsx** *(real)*: Tolerance for integer values when loading relaxed solution

This tolerance is used to distinguish integer variables that are set to an integer value by the user, or integer variables that are fractional. See the option relaxed.

Default: $1.0e^{-3}$

**feaspump** *(integer)*: Whether to run the Feasibility Pump

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not run the Feasibility Pump</td>
</tr>
<tr>
<td>1</td>
<td>Run the Feasibility Pump if there are no semicontinuous or semiinteger variables and no special ordered sets</td>
</tr>
<tr>
<td>2</td>
<td>Always run the Feasibility Pump</td>
</tr>
</tbody>
</table>

**fp_acttol** *(real)*: Tolerance on when to consider an equation as active
**fp_cutoffdecr** *(real)*: Additional relative decrement of cutoff value for the original objective function

Default: 0.1

**fp_dumpsprob** *(boolean)*: Whether to dump subproblems (NLPs and MIPs) into GAMS files

Default: 0

**fp_integercuts** *(integer)*: Whether to add integer cuts after an NLP subproblem has been solved to optimality

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use integer cuts</td>
</tr>
<tr>
<td>1</td>
<td>Use integer cuts only for mixed-binary problems</td>
</tr>
<tr>
<td>2</td>
<td>Always use integer cuts</td>
</tr>
</tbody>
</table>

**fp_iterlimit** *(integer)*: Major Iteration limit

Default: 20

**fp_mipgap** *(real)*: Optimality tolerance (relative gap) to use for solving MIP projection problem

Default: 0.01

**fp_projcuts** *(boolean)*: Whether to add cut derived from projection of MIP solution onto NLP feasible set

Default: 1

**fp_projectrotol** *(real)*: Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP

Default: 1E-4

**fp_sollimit** *(integer)*: Stop when a number of (improving) solutions has been bound

Default: maxint

**fp_stalllimit** *(integer)*: Stop when no improving solution has been bound for a number of FP iterations, -1 to disable

Range: {-1, ..., ∞}

Default: 5

**fp_subsolverlog** *(boolean)*: Whether to show the log of subsolvers

Default: 0

**fp_timelimit** *(real)*: Time limit

Default: maxdouble
**fp_transfercuts** *(boolean)*: Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible) ←

Default: 1

**infbnd** *(real)*: Bound to use for unbounded integer variables in integer cuts ←

Value to use for missing bounds on discrete variables when constructing integer cuts.

Default: 10000

**infeasder** *(integer)*: Use derivatives of infeasible nonlinear subproblems ←

This option is to determine whether linearizations of infeasible NLP subproblems are added or not added to the MIP master problem.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No linearizations of infeasible NLP subproblems. This is the default option in which no linearizations are added in the infeasible NLP subproblems. In this case a simple integer cut is added to remove from consideration the 0-1 vector that gave rise to the infeasible NLP. Since this may slow the convergence, it is recommended to reformulate the MINLP with &quot;elastic&quot; constraints (i.e., adding slacks to infeasible constraints and adding a penalty for them in the objective) to ensure that the NLP subproblems are mathematically feasible.</td>
</tr>
<tr>
<td>1</td>
<td>Add linearization for infeasible NLP subproblems. This will add linearizations derived from the infeasible NLP subproblem to the master problem. This option is recommended to speed up convergence when the MINLP is known to be convex (i.e. its continuous relaxation is convex). The possibility of cutting-off the global optimum is increased if it is used for a nonconvex MINLP.</td>
</tr>
</tbody>
</table>

**maxcycles** *(integer)*: Maximum number of cycles ←

The maximum number of cycles or major iterations performed by DICOPT.

Default: 20

**mipiterlim** *(string)*: List of iteration limits ←

`mipiterlim i1 i2 ... in` sets an iteration limit on individual MIP master problems. The last number `in` is valid for all subsequent cycles `n, n + 1, ...`. A number of `−1` indicates that there is no (individual) limit on the corresponding MIP master problem. A global iteration limit is maintained through the GAMS option `iterlim`.

Example: `mipiterlim 10000 -1`

The first MIP master problem cannot use more than 10000 iterations, while subsequent MIP master problems are not individually restricted.

Example: `mipiterlim 10000`

Sets an iteration limit of 10000 on all MIP master problems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict iteration counts on individual solves of MIP master problems. The default for this option is not to restrict iteration counts on individual solves of MIP master problems.
mipoptfile (string): List of option files for MIP solver

mipoptfile $s_1 \ s_2 \ldots \ s_n$ specifies the option file to be used for the MIP master problems. Several option files can be specified, separated by a blank. If a digit 1 is entered the default option file for the MIP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent MIP master problems.

Example: \texttt{mipoptfile mip.opt mip2.opt 0}  
This option will cause the first MIP master problem solver to read the option file \texttt{mip.opt} and the second one to read the option file \texttt{mip2.opt}; subsequent MIP master problem solvers will not use any option file.

Example: \texttt{mipoptfile 1}  
This will cause the MIP solver for all MIP subproblems to read a default option file (e.g. \texttt{cplex.opt}, \texttt{xpress.opt}, \texttt{gurobi.opt} etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

mipreslim (string): List of resource limits

mipreslim $x_1 \ x_2 \ldots \ x_n$ sets a resource (time) limit on individual MIP master problems. The last number $x_n$ is valid for all subsequent cycles $n, n+1, \ldots$. A number $-1.0$ means that the corresponding MIP master problem is not individually time restricted. A global time limit is maintained through the GAMS option \texttt{reslim}.

Example: \texttt{mipreslim -1 10000 -1}  
The MIP master problem in cycle 2 cannot use more than 100 seconds, while subsequent MIP master problems are not individually restricted.

Example: \texttt{mipreslim 1000}  
Sets a time limit on all MIP master problems of 1000 seconds.

When this option is used it is advised to have the option \texttt{continue} set to its default of 2. The default for this option is not to restrict individually the time a solver can spent on the MIP master problem.

mipsolver (string): List of MIP solvers

This option specifies with MIP solver to use for the MIP master problems.

Example: \texttt{mipsolver cplex xpress}  
This instructs \texttt{DICOPT} to use Cplex for the first MIP and XPRESS for the second and subsequent MIP problems. The last entry may be used for more than one problem.

The names to be used for the solvers are the same as one uses in the GAMS statement \texttt{OPTION MIP=...}; The default is to use the default MIP solver. Note that changing from one MIP solver to another can lead to different results, and may cause \texttt{DICOPT} to follow a different path.

nlpiterlim (string): List of iteration limits

nlpiterlim $i_1 \ i_2 \ldots \ i_n$ sets an iteration limit on individual NLP subproblems. The last number $i_n$ is valid for all subsequent cycles $n, n+1, \ldots$. A number of $-1$ indicates that there is no (individual) limit on the corresponding NLP subproblem. A global iteration limit is maintained through the GAMS option \texttt{reslim}. 
Example: `nlpiterlim 1000 -1`  
The first (relaxed) NLP subproblem cannot use more than 1000 iterations, while subsequent NLP subproblems are not individually restricted.

Example: `nlpiterlim 1000`  
Sets an iteration limit of 1000 on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. This default does not restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

**nlpoptfile** *(string)*: List of option files for NLP solver  

`nlpoptfile s_1 s_2 ... s_n` specifies the option file to be used for the NLP subproblems. Several option files can be specified, separated by a blank. If a digit 1 is entered, the default option file for the NLP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent NLP subproblems.

Example: `nlpoptfile nlp.opt nlp2.opt 0`  
This option will cause the first NLP subproblem solver to read the option file `nlp.opt` and the second one to read the option file `nlp2.opt`; subsequent NLP subproblem solvers will not use any option file.

Example: `nlpoptfile 1`  
This will cause the NLP solver for all NLP subproblems to read a default option file (e.g. `conopt.opt`, `minos.opt`, `snopt.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

**nlpreslim** *(string)*: List of resource limits  

`nlpreslim x_1 x_2 ... x_n` sets a resource (time) limit on individual NLP subproblems. The last number `x_n` is valid for all subsequent cycles `n, n+1, ...`. A number `-1.0` means that the corresponding NLP subproblem is not individually time restricted. A global time limit is maintained through the GAMS option `reslim`.

Example: `nlpreslim 100 -1`  
The first (relaxed) NLP subproblem can not use more than 100 seconds, while subsequent NLP subproblems are not individually restricted.

Example: `nlpreslim 1000`  
Sets a time limit of 1000 seconds on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. This default does not restrict the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

**nlpsolver** *(string)*: List of NLP solvers  

`nlpsolver s_1 s_2 ... s_n`. This option specifies which NLP solver to use for the NLP subproblems.

Example: `nlpsolver conopt minos snopt`  
tells DICOPT to use CONOPT for the relaxed NLP, MINOS for the second NLP subproblem, and SNOPT for the third and subsequent ones. The last entry is used for more than one subproblem: for all subsequent ones DICOPT will use the last specified solver.
The names to be used for the solvers are the same as those used in the GAMS statement `OPTION NLP=...;`. The default is to use the default NLP solver. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

**nlptracefile (string):** Base name of trace files

Name of the files written if the option `nlptracelevel` is set. Only the stem is needed: if the name is specified as `nlptracefile nlptrace`, then files of the form `nlptrace.001`, `nlptrace.002`, etc. are written. These files contain the settings of the integer variables so that NLP subproblems can be investigated independently of DICOPT.

Default: `nlptrace`

**nlptracelevel (integer):** Trace level

This sets the level for NLP tracing, which writes a file for each NLP sub-problem, so that NLP sub-problems can be investigated outside the DICOPT environment. See also the option `DICOPT.nlptracefile "nlptracefile"`. By including a trace file in your original problem and changing it into an MINLP problem, the subproblem will be solved directly by an NLP solver. This option only works if the names in the model (names of variables and equations) are exported by GAMS. This can be accomplished by using the `m.dictfile` model suffix, as in `m.dictfile=1;`. In general it is more convenient to use the CONVERT solver to generate isolated NLP models (see section Model Debugging).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
</table>
| 0     | No trace info is written  
No trace files are written. This is the default. |
| 1     | GAMS file with fixed integer variables  
A GAMS file for each NLP subproblem is written which fixes the discrete variables. |
| 2     | Include levels of continuous variables  
As `nlptracelevel 1`, but in addition level values of the continuous variables are written. |
| 3     | Include all levels and marginals  
As `nlptracelevel 2`, but in addition marginal values for the equations and variables are written. |

**optca (string):** List of OPTCA values

`optca x_1 x_2 ... x_n`. The absolute optimality criterion for the MIP master problems. The GAMS option `optca` is ignored, as, by default, DICOPT wants to solve MIP master problems to optimality. It is possible to stop the MIP solver earlier to allow it to solve a large problem, by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optca`, the MIP solver is instructed to stop as soon as the gap between the best possible integer solution and the best found integer solution is less than `x`, i.e. stop as soon as

$$|\text{BestFound} - \text{BestPossible}| \leq x$$

It is possible to specify a different `optca` value for each cycle. The last number `x_n` is valid for all subsequent cycles `n, n+1, ...`

Example: `optca 10`

Stop the search in all MIP problems as soon as the absolute gap is less than 10.
Example: optca 0 10 0
Sets a nonzero optca value of 10 for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

optcr (string): List of OPTCR values ←

optcr x_1 x_2 \ldots x_n. The relative optimality criterion for the MIP master problems. The GAMS option optcr is ignored, as by default DICOPT wants to solve MIP master problems to optimality. To allow it to solve a large problem it is possible to stop the MIP solver earlier by specifying a value for optca or optcr in a DICOPT option file. With setting a value for optcr, the MIP solver is instructed to stop as soon as the relative gap between the best possible integer solution and the best found integer solution is less than x, i.e., stop as soon as

\[
\frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \leq x
\]

Note that the relative gap cannot be evaluated if the best possible integer solution is zero. In these cases the absolute optimality criterion optca can be used. It is possible to specify a different optcr value for each cycle. The last number x_n is valid for all subsequent cycles n, n + 1, \ldots.

Example: optcr 0.1
Stop the search in all the MIP problems as soon as the relative gap is smaller than 10%.

Example: optcr 0 0.01 0
Sets a nonzero optcr value of 1% for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

relaxed (integer): How to start DICOPT ←

In some cases it may be possible to use a known configuration of the discrete variables. Some users have very difficult problems, where the relaxed problem cannot be solved but where NLP sub-problems with the integer variables fixed are much easier. If a reasonable integer configuration is known in advance in these cases we can bypass the relaxed NLP and tell DICOPT to directly start with this integer configuration. The integer variables need to be specified by the user before the solve statement by assigning values to the levels, as in \texttt{Y.L(I) = INITVAL(I);}.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Start with all integers fixed to the starting value</td>
</tr>
<tr>
<td></td>
<td>The first NLP sub-problem will be executed with all integer variables fixed to the values specified by the user. If you don’t assign a value to an integer variable, it will retain its current value, which is zero by default.</td>
</tr>
<tr>
<td>1</td>
<td>Start with relaxed NLP</td>
</tr>
<tr>
<td></td>
<td>The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.</td>
</tr>
<tr>
<td>2</td>
<td>Start with mixture of fixed and relaxed integers</td>
</tr>
<tr>
<td></td>
<td>The first NLP subproblem will be executed with some variables fixed and some relaxed. The program distinguishes the fixed from the relaxed variables by comparing the initial values against the bounds and the tolerance allowed \texttt{EPSX}. \texttt{EPSX} has a default value of 1.e-3. This can be changed through the option file.</td>
</tr>
</tbody>
</table>
**solvelink** *(integer)*: Solvelink for NLP and MIP subsolver

This option defines the solvelink used for the NLP and MIP subsolver.

Default: 5

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Call NLP and MIP solver via script</td>
</tr>
<tr>
<td>2</td>
<td>Call NLP and MIP solver via module</td>
</tr>
<tr>
<td>5</td>
<td>Call NLP and MIP solver in memory</td>
</tr>
</tbody>
</table>

**stop** *(integer)*: Stopping criterion

This option defines the stopping criterion to be used. The search is always stopped when the (minor) iteration limit (the *iterlim* option), the resource limit (the *reslim* option), or the major iteration limit (see *maxcycles*) is hit or when the MIP master problem becomes infeasible.

Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section Modeling shows some experiments with these stopping criteria.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stop on maxcycles</td>
</tr>
<tr>
<td>1</td>
<td>Stop on crossover</td>
</tr>
<tr>
<td>2</td>
<td>Stop on worsening</td>
</tr>
<tr>
<td>3</td>
<td>Stop on crossover or worsening</td>
</tr>
</tbody>
</table>

- **Stop on maxcycles**
  Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will often find the optimal solution using one of the more optimistic stopping rules. Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will find often the optimal solution using one of the more optimistic stopping rules.

- **Stop on crossover**
  Stop as soon as the bound defined by the objective of the last MIP master problem is worse or close (w.r.t. GAMS option *optcr*) to the best NLP solution found (a crossover occurred). For convex problems this gives a global solution, provided the weights are large enough and *optcr* is set to 0. This stopping criterion should only be used if it is known or it is very likely that the nonlinear functions are convex. In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut off with the setting *stop* 1.

- **Stop on worsening**
  Stop as soon as the NLP subproblems stop improving. This worsening criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic often works very well. Even on convex problems, in many cases it terminates the search very early while providing an optimal or a very good integer solution. The criterion is not checked before major iteration three.

- **Stop on crossover or worsening**
  Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).
5.14 DICOPT

usexinit (boolean): Use the user initial point as starting point for all NLP solves

Default: 0

weight (real): Penalty parameter, set above 1e20 to disable penalty relaxation

The value of the penalty coefficients.

Default: 1000

5.14.9 DICOPT Output

DICOPT generates lots of output on the screen. DICOPT itself and also the NLP and MIP solvers that handle the sub-problems write messages to the screen. The most important part is the last part of the screen output.

In this section we will discuss the output DICOPT writes to the screen and the listing file using the model procsel.gms (this model is part of the GAMS model library). A DICOPT log is written and the reason why DICOPT terminated is explained.

--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:

<table>
<thead>
<tr>
<th>Step</th>
<th>Iter</th>
<th>Objective</th>
<th>CPU time (Sec)</th>
<th>Iterations</th>
<th>Evaluation Errors</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP  1</td>
<td>5</td>
<td>5.35021</td>
<td>0.05</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP  1</td>
<td>2.48869</td>
<td>0.28</td>
<td>7</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP  2</td>
<td>1.72097&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP  2</td>
<td>2.17864</td>
<td>0.22</td>
<td>10</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP  3</td>
<td>1.92310&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP  3</td>
<td>1.42129</td>
<td>0.22</td>
<td>12</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP  4</td>
<td>1.41100</td>
<td>0.00</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
</tbody>
</table>

--- DICOPT: Terminating...
--- DICOPT: Stopped on NLP worsening

The search was stopped because the objective function of the NLP subproblems started to deteriorate.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCESL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion

Notice that the integer solutions are provided by the NLP's except for major iteration one (the first NLP is the relaxed NLP). For all NLP's except the relaxed one the binary variables are fixed, according to a pattern determined by the previous MIP which operates on a linearized model. The integer solutions marked with a '<' are an improvement. We see that the NLP in cycle 4 starts to deteriorate, and DICOPT stops based on its default stopping rule.

Note that if the criterion stop 1 had been used the search would have been terminated at iteration 3. The reason is that the upper bound to the profit predicted by the MIP (1.42129) exceeds the best current NLP solution (1.9231). Since it can be shown that the MINLP involves convex nonlinear functions, 1.9231 is the global optimum and the criterion stop 1 is rigorous.

A similar output can be found in the listing file:
**SOLVE SUMMARY**

<table>
<thead>
<tr>
<th>MODEL</th>
<th>OBJECTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>pr</td>
</tr>
<tr>
<td>TYPE</td>
<td>MINLP</td>
</tr>
<tr>
<td>DIRECTION</td>
<td>MAXIMIZE</td>
</tr>
<tr>
<td>SOLVER</td>
<td>DICOPT</td>
</tr>
<tr>
<td>FROM LINE</td>
<td>98</td>
</tr>
</tbody>
</table>

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 8 INTEGER SOLUTION
**** OBJECTIVE VALUE 1.9231

RESOURCE USAGE, LIMIT 0.771 1000.000
ITERATION COUNT, LIMIT 51 10000
EVALUATION ERRORS 0 0

--- DICOPT: Stopped on NLP worsening

The search was stopped because the objective function of the NLP subproblems started to deteriorate.

---

---

Dicopt2x-C Jul 4, 2001 WIN.DI.DI 20.1 026.020.039.WAT

---

Aldo Vecchietti and Ignacio E. Grossmann
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Erwin Kalvelagen
GAMS Development Corp.
1217 Potomac Street, N.W.
Washington DC 20007

---

DICOPT Log File

---

Major Major Objective CPU time Itera- Evaluation Solver
Step Iter Function (Sec) tions Errors
NLP 1 5.35021 0.05 8 0 conopt
MIP 1 2.48869 0.28 7 0 cplex
NLP 2 1.72097< 0.00 3 0 conopt
MIP 2 2.17864 0.22 10 0 cplex
NLP 3 1.92310< 0.00 3 0 conopt
MIP 3 1.42129 0.22 12 0 cplex
NLP 4 1.41100 0.00 8 0 conopt

---

Total solver times : NLP = 0.05 MIP = 0.72
Perc. of total : NLP = 6.59 MIP = 93.41

---

In case the DICOPT run was not successful, or if one of the subproblems could not be solved, the listing file will contain all the status information provided by the solvers of the subproblems. For each iteration the configuration of the binary variables will also be printed. This extra information can also be requested via the GAMS option:

```
option sysout = on;
```
5.14.10 Special Notes

This section covers some special topics of interest to users of DICOPT.

5.14.10.1 Stopping Rule

Although the default stopping rule behaves quite well in practice there some cases where it terminates too early. In this section we discuss the use of the stopping criteria.

When we run the example procsel.gms with stopping criterion 0, we see the following DICOPT log:

--- DICOPT: Starting major iteration 10
--- DICOPT: Search terminated: infeasible MIP master problem
--- DICOPT: Log File:

<table>
<thead>
<tr>
<th>Step</th>
<th>Major Objective</th>
<th>Function</th>
<th>CPU time</th>
<th>Iterations</th>
<th>Evaluation</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP 1</td>
<td>5.35021</td>
<td>0.06</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 1</td>
<td>2.48869</td>
<td>0.16</td>
<td>7</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 2</td>
<td>1.72097&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 2</td>
<td>2.17864</td>
<td>0.10</td>
<td>10</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 3</td>
<td>1.92310&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 3</td>
<td>1.42129</td>
<td>0.11</td>
<td>12</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 4</td>
<td>1.41100</td>
<td>0.00</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 4</td>
<td>0.00000</td>
<td>0.22</td>
<td>23</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 5</td>
<td>0.00000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 5</td>
<td>-0.27778</td>
<td>0.16</td>
<td>22</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 6</td>
<td>-0.27778</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 6</td>
<td>-1.00000</td>
<td>0.16</td>
<td>21</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 7</td>
<td>-1.00000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 7</td>
<td>-1.50000</td>
<td>0.22</td>
<td>16</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 8</td>
<td>-1.50000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 8</td>
<td>-2.50000</td>
<td>0.11</td>
<td>16</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
<tr>
<td>NLP 9</td>
<td>-2.50000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
<td></td>
</tr>
<tr>
<td>MIP 9</td>
<td><em>Infeas</em></td>
<td>0.11</td>
<td>0</td>
<td>0</td>
<td>cplex</td>
<td></td>
</tr>
</tbody>
</table>

--- DICOPT: Terminating...
--- DICOPT: Stopped on infeasible MIP

The search was stopped because the last MIP problem was infeasible. DICOPT will not be able to find a better integer solution.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- procsel.gms(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion

This example shows some behavioral features that are not uncommon for other MINLP models. First, DICOPT often finds the best integer solution in the first few major iterations. Second, in many cases as soon as the NLP's start to give worse integer solution no better integer solution will be found. This observation is the motivation to make stopping option 2, where DICOPT stops as soon as the NLP's start to deteriorate, the default stopping rule. In this example DICOPT would have stopped in major iteration 4 (you can verify this in the previous section). In many cases this will give the best integer solution. For this problem, DICOPT has indeed found the global optimum.
Based on experience with other models, we find that the default stopping rule (stop when the NLP becomes worse) performs well in practice. In many cases it finds the global optimum solution for both convex and non-convex problems. In some cases, however, it may provide a sub-optimal solution. In those cases where you want more reassurance that no good integer solutions are missed you can use one of the other stopping rules.

Changing the MIP or NLP solver can change the path that DICOPT follows, since the sub-problems may have non-unique solutions. The optimum stopping rule for a particular problem depends on the MIP and NLP solvers used.

The bounds of the MIP master problem are not rigorous in the case of non-convex problems. Therefore, the global optimum can be cut-off with \texttt{stop 1}. However, this option is the best stopping criterion for convex problems.

5.14.10.2 Solving the NLP Problems

Using a combination of NLP solvers has been found effective in cases where the relaxed NLP and/or the other NLP sub-problems are very difficult. For example, MINOS has many more difficulties to establish if a model is infeasible, so one would like to use CONOPT for NLP subproblems that are either infeasible or barely feasible. The \texttt{nlpsolver} option can be used to specify the NLP solver to be used for each iteration.

Infeasible NLP sub-problems can be problematic for DICOPT. Those subproblems cannot be used to form a new linearization. Effectively only the current integer configuration is excluded from further consideration by adding appropriate integer cuts, but otherwise an infeasible NLP sub-problem provides no useful information to be used by the DICOPT algorithm. If your model shows many infeasible NLP sub-problems you can try to use the \texttt{infeasder} option. Otherwise a strategy that can help is to introduce explicit slack variables and add them with a penalty to the objective function.

Assume your model is of the form:

\[
\begin{align*}
\min & \quad f(x, y) \\
& \quad g(x, y) \sim b \\
& \quad \ell \leq x \leq u \\
& \quad y \in \{0, 1\}
\end{align*}
\]

where \(\sim\) is a vector of relational operators \(\{\leq, =, \geq\}\). \(x\) are continuous variables and \(y\) are the binary variables. If many of the NLP subproblems are infeasible, we can try the following \texttt{elastic} formulation:

\[
\begin{align*}
\min & \quad f(x, y) + M \sum_i (s_i^+ + s_i^-) \\
& \quad y = y^B + s^+ - s^- \\
& \quad g(x, y) \sim b \\
& \quad \ell \leq x \leq u \\
& \quad 0 \leq y \leq 1 \\
& \quad 0 \leq s^+, s^- \leq 1 \\
& \quad y^B \in \{0, 1\}
\end{align*}
\]

I.e., the variables \(y\) are relaxed to be continuous with bounds \([0, 1]\), and binary variables \(y^B\) are introduced that are related to the variables \(y\) through a set of the slack variables \(s^+, s^-\). The slack variables are added to the objective with a penalty parameter \(M\). The choice of a value for \(M\) depends on the size of \(f(x, y)\), on the behavior of the model, etc. Typical values are 100 or 1000.
5.14.10.3 Solving the MIP Master Problems

MIP master problems may become expensive to solve when there are many discrete variables. One of the first things to try is to see if a different MIP solver can solve your particular problems more efficiently.

Different formulations can have dramatic impact on the performance of MIP solvers. Therefore it is advised to try out several alternative formulations. The use of priorities can have a big impact on some models. It is possible to specify a nonzero value for `OPTCA` and `OPTCR` in order to prevent the MIP solver from spending an unacceptable long time proving optimality of MIP master problems.

If the MIP master problem is infeasible the `DICOPT` solver will terminate. In this case you may want to try the same reformulation discussed in the previous paragraph.

5.14.10.4 Feasibility Pump

The feasibility pump is similar to the Outer-approximation, but its objective is to completely focus on finding good feasible solutions rather than optimal ones. The main idea of this algorithm is to decompose the original mathematical programming problem in two parts: integer feasibility and constraint feasibility. By solving an MIP subproblem its solution is an integer feasible solution, which may violate the constraints; and by solving a continuous relaxation of the original MINLP (NLP subproblem) the solution is constraint feasible but might not be integral. By minimizing in successive iterations the distance between these two types of solutions it is expected to achieve a solution that is both constraint and integral feasible.

The feasibility pump can be used as a standalone solver for convex MINLP problems. This is achieved by iteratively applying the method, while including a bound to the objective function. This bound is obtained by the best known solution and an epsilon improvement. This will result in the global optimum of a convex MINLP [28]. The drawback of this algorithm is that it may require many iterations, since each time the objective function is restricted to improve only by epsilon.

In DICOPT, the feasibility pump can be applied before the Outer-approximation method by setting option `feaspump`. In the feasibility pump, large improvements in the objective function are enforced at each iteration (option `fp_cutoffdecr`). After the method finishes, all the cuts and the best known solution are passed to the Outer-approximation method to prove optimality. More details on the implementation of the feasibility pump in DICOPT can be found in [19].

5.14.10.5 Model Debugging

In this paragraph we discuss a few techniques that can be helpful in debugging your MINLP model.

- Start with solving the model as an `RMINLP` model. Make sure this model solves reliably before solving it as a proper `MINLP` model. If you have access to different NLP solvers, make sure the `RMINLP` model solves smoothly with all NLP solvers. `CONOPT`, especially, can generate useful diagnostics such as Jacobian elements (i.e. matrix elements) that become too large.
- Try different NLP and MIP solvers on the subproblems. Example: use the GAMS statement `OPTION NLP=KNITRO;` to solve all NLP subproblem using the solver `KNITRO`.
- The GAMS option statement `OPTION SYSOUT = ON;` can generate extra solver information that can be helpful for diagnosing problems.
- If many of the NLP subproblems are infeasible, add slacks as described in section `Solving the NLP Problems`.
- Run `DICOPT` in pedantic mode by using the `DICOPT` option: `CONTINUE 0`. Make sure all NLP subproblems solve to optimality.
• Don't allow any nonlinear function evaluation errors, i.e. keep the DOMLIM limit at zero. See the discussion on DOMLIM in section The OPTION Statement.

• If you have access to another MINLP solver such as AlphaECP, SHOT, or SBB or even global solvers like Antigone or BARON, try to use a different solver on your model. To select another solver (here SBB) use the following GAMS option statement: OPTION MINLP=SBB;.

• Individual NLP or MIP subproblems can be extracted from the MINLP by using the CONVERT solver, which will write a model in scalar GAMS notation that can then be solved using any GAMS NLP or MIP solver. E.g., to generate the second NLP subproblem, you can use the following DICOPT option: NLPSOLVER CONOPT CONVERT. The model will be written to the file GAMS.GMS. A disadvantage of this technique is that some precision is lost due to the fact that files are being written in plain ASCII. The advantage is that you can visually inspect these files and look for possible problems such as poor scaling.

5.15 EXAMINER

5.15.1 Introduction

This document describes GAMS/Examiner, a tool for examining points and making an unbiased, independent assessment of their merit. In short, it checks if solutions are really solutions. As an example, it can take a solution point reported as optimal by a solver and examine it for primal feasibility, dual feasibility, and optimality. Examiner has a number of different modes, allowing it to check the input point from GAMS/Base as well as the solution passed by a solver back to GAMS.

Many of the tests done by Examiner (perhaps all of them!) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, the ability to make an independent, transparent check of a solver's solution is very useful in solver development, testing, and debugging. It is also useful when comparing the solutions returned by two different solvers. Finally, a tool like the Examiner allows one to examine solutions using different optimality tolerances and optimality criteria in a way that is not possible when working with the solvers directly.

GAMS/Examiner is installed automatically with your GAMS system. Without a full GAMS/Base license, examiner will run in student or demonstration mode (i.e. it will examine small models only).

5.15.2 Usage

Examiner can be used with all supported model types. Since Examiner doesn't really solve any problems, it is not a good choice for a default solver, and when installing GAMS it does not appear as an option in the list of possible solver defaults. However, you can choose Examiner via the command line:

    gams transport LP=examiner;

or via a GAMS option statement

    option LP=examiner;
somewhere before the solve statement.

Since Examiner is not really a solver, many of the usual GAMS options controlling solvers have no impact on it. However, the sysout option is interpreted in the usual way.

The optimality checks done in Examiner are first-order optimality checks done at a given point. A discussion here of these conditions and all they imply would be redundant: any good intro text in optimization will cover them. For linear programming, first-order optimality is all one needs to prove global optimality. For nonlinear programming, these conditions may or may not be necessary or sufficient for optimality; this depends on the convexity of the feasible set and objective and the form of the constraints. For integer programming models, these checks only make sense if we turn the global problem into a local one by adding bounds to the model, essentially fixing each discrete variable to its current value: these bounds are added automatically by Examiner.

Examiner runs in two basic modes of operation: it can examine the input point passed from GAMS/Base to the solver, and it can examine the point passed from the solver back to GAMS. Each mode can be used independent of the other. By default it will operate in the first mode, examining the initial “solution” passed to it by GAMS, but only if GAMS indicates it is passing an advanced basis to the solver (cf. the GAMS User Guide and the bratio option). If you wish to use the second solver-check mode, you may specify an appropriate subsolver using the subsolver option (see section Options). If no subsolver is selected, the default solver for the model type being solved is used. In most cases you will want to use an option file to specify exactly what type of examination you wish to perform. The rules for using an option file are described in The Solver Options File.

5.15.2.1 Solution Points: Definition

There are a number of different ways a solution point can be defined. Of course the different definitions will typically result in the same points being produced, but there are cases where this will not be precisely so. Since Examiner is intended to explore and analyze these cases, we must make these definitions precise. The following four points are defined and used in Examiner:

1. The gamspoint is the input point provided by GAMS to Examiner. The GAMS input point includes level & marginal values for the rows and columns: Examiner uses these exactly as given.

2. The initpoint is determined by the variable levels (primal vars) and equation marginals (dual vars) provided by GAMS to Examiner. These values are used to compute the equation levels and variable marginals / reduced costs using the function evaluator in Examiner, rather than using the values passed in by GAMS.

3. The solupoint is similar to the initpoint: it uses the variable levels (primal vars) and equation marginals (dual vars) to compute the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.

4. The solvpoint is the point returned by the subsolver. The subsolver returns both level and marginal values for the rows and columns: Examiner uses these, exactly as given.

5.15.2.2 Checks Performed

There are a number of checks that can be performed on any of the solution points. By default, Examiner tries to choose the appropriate checks. For example, if a primal simplex solver returns a model status of nonoptimal, the only checks that make sense are feasibility in the primal variables and constraints. However, this automatic choice of appropriate checks is not possible when checking points passed in from GAMS/Base.

1. Primal variable feasibility: check that all primal variables are within bounds.
2. **Primal constraint feasibility**: check that all primal constraints are satisfied.

3. **Dual variable feasibility**: check that all dual variables are within bounds.

4. **Dual constraint feasibility**: check that all dual constraints are satisfied.

5. **Primal complementary slackness**: check complementarity between the primal variables and the dual constraints / reduced costs.

6. **Dual complementary slackness**: check complementarity between the dual variables / equation marginals and the equation slacks.

7. **Equilibrium condition complementarity**: check complementarity of the equation/variable pairs in complementarity models (MCP, MPEC).

The checks above are implemented with default tolerances. These tolerances can be changed via an option file (see section Options).

Different ways exist to check the items mentioned above. For example, different norms can be used to measure the error of the residual when checking for primal feasibility. Currently, we have only implemented one way to make these checks.

For optimization models, it is possible to define a merit function that measures how well a given point satisfies conditions 1 through 6 above. Such a function is always nonnegative, and its value represents the error between the given point and a true solution. If such an input point is well-defined (e.g. if only one point is selected for examination) then Examiner will compute the merit function value at this point and return it in the RObj model attribute.

### 5.15.2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, however, it is important to take scaling into account. Consider the effect of row scaling on the simple constraint $x^2 \leq 9$ where $x = 3.5$. Multiplying this constraint through by large or small constants changes the amount of the constraint violation proportionately, but the distance to feasibility is not changed. Applying row scaling to the original model eliminates this problem.

Most solvers scale a model before solving it, so any feasibility or optimality checks and tolerances are applied to the scaled model. The process of unscaling the model can result in a loss of feasibility or optimality. Even though we do not have access to the scales applied by the solver and cannot precisely construct the same scaled model, we can get a better idea of how the solver performed by looking at a model scaled by Examiner than by looking at the original.

It is also interesting to see what the model scaling looks like, even if we do not apply the scales to do the Examiner checks. If the row scales are in a nice range, say $[.1,100]$, we can have some confidence that the model is well-scaled. In contrast, if the row scales are in the range $[1,1e8]$ we may question the precision of the solution provided.

For each row, Examiner computes the true row scale as

$$\max_j (\|RH_{ij}\|, \max_j (\|A_{ij}\| \cdot \max (1, \|x_j\|)))$$

In this way variables with a large level value lead to large scale factors. To make the scale factor independent of the variable values, use an option file line of "AbsXScale 0". This replaces the term $\max (1, \|x_j\|)$ above with 1.

Since the user may wish to limit the size of the scale factors applied, the true row scales are projected onto the scale factor bounds to get the applied scale factors. The scale factors are applied when making a scaled check by dividing the rows by the scale factors and multiplying the corresponding Lagrange multipliers by these same factors. When making unscaled checks information about the true scales is still included in the output to give the user a hint about potential scaling issues.

Note that the scaled and unscaled checks are made independently. By default only the unscaled checks are performed. If you turn the scaled checks on via an option file line "scaled 1", this will not turn off the unscaled checks. You will need an option file line of "unscaled 0" to turn off unscaled checks.
### 5.15.3 Options

The following options control the behavior of GAMS/Examiner. Many of these are boolean (i.e. on/off) options. In this case, zero indicates off, nonzero on. For details on how to create and use an option file, see the section on the Solver Option File.

#### 5.15.3.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>absXScale</td>
<td>Whether to make scale factors dependent on x values.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>If on, the matrix coefficients are multiplied by max(1,abs(x)) when computing the scale factors. If off, the matrix coefficients are taken as is. See Section Scaling.</td>
<td></td>
</tr>
<tr>
<td>dumpGamsPoint</td>
<td>Whether to dump the GamsPoint to a basis file in GAMS source format.</td>
<td>0</td>
</tr>
<tr>
<td>dumpInitPoint</td>
<td>Whether to dump the InitPoint to a basis file in GAMS source format.</td>
<td>0</td>
</tr>
<tr>
<td>dumpSoluPoint</td>
<td>Whether to dump the SoluPoint to a basis file in GAMS source format.</td>
<td>0</td>
</tr>
<tr>
<td>dumpSolvPoint</td>
<td>Whether to dump the SolvPoint to a basis file in GAMS source format.</td>
<td>0</td>
</tr>
<tr>
<td>examineGamsPoint</td>
<td>Whether to examine the GamsPoint.</td>
<td>0</td>
</tr>
<tr>
<td>examineInitPoint</td>
<td>Whether to examine the InitPoint.</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise.</td>
<td></td>
</tr>
<tr>
<td>examineSoluPoint</td>
<td>Whether to examine the SoluPoint.</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>By default, this option is on if a subsolver has been selected, and off otherwise.</td>
<td></td>
</tr>
<tr>
<td>examineSolvPoint</td>
<td>Whether to examine the SolvPoint.</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>By default, this option is on if a subsolver has been selected, and off otherwise.</td>
<td></td>
</tr>
<tr>
<td>fCheckAll</td>
<td>If set, forces all checks on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckATTR</td>
<td>If set, forces the model attributes check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckDCMP</td>
<td>If set, forces the dual complementary slackness check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckDCON</td>
<td>If set, forces the dual constraint feasibility check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckDVAR</td>
<td>If set, forces the dual variable feasibility check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckPCMP</td>
<td>If set, forces the primal complementary slackness check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckPCON</td>
<td>If set, forces the primal constraint feasibility check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>fCheckPVAR</td>
<td>If set, forces the primal variable feasibility check on or off.</td>
<td>auto</td>
</tr>
<tr>
<td>objvAutoAdjust</td>
<td>Adjust objective variable to satisfy objective equation</td>
<td>0</td>
</tr>
<tr>
<td>perpSys</td>
<td>Controls output during examination of solution points.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If on, print out the point in a way that allows for easy visual inspection and verification of the KKT or first order optimality conditions. First, the primal level values and bounds are printed next to the reduced costs. Next, the duals levels and bounds are printed next to the row slacks.</td>
<td></td>
</tr>
<tr>
<td>returnGamsPoint</td>
<td>Whether to return the GamsPoint as a solution to GAMS/Base.</td>
<td>0</td>
</tr>
<tr>
<td>returnInitPoint</td>
<td>Whether to return the InitPoint as a solution to GAMS/Base.</td>
<td>auto</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>returnSoluPoint</td>
<td>Whether to return the SoluPoint as a solution to GAMS/Base.</td>
<td>auto</td>
</tr>
<tr>
<td>returnSolvPoint</td>
<td>Whether to return the SolvPoint as a solution to GAMS/Base.</td>
<td>auto</td>
</tr>
<tr>
<td>scaled</td>
<td>Whether to apply checks to a scaled version of the model.</td>
<td>0</td>
</tr>
<tr>
<td>scaleLB</td>
<td>Lower bound for applied row scales.</td>
<td>1</td>
</tr>
<tr>
<td>scaleUB</td>
<td>Upper bound for applied row scales.</td>
<td>maxdouble</td>
</tr>
<tr>
<td>showSlacks</td>
<td>explicitly show the infeasibilities or slacks for failed checks</td>
<td>0</td>
</tr>
<tr>
<td>subSolver</td>
<td>Indicates what subsolver to run.</td>
<td>auto</td>
</tr>
<tr>
<td>subSolverOpt</td>
<td>optfile value to pass to the subsolver</td>
<td>auto</td>
</tr>
<tr>
<td>trace</td>
<td>If set, trace information will be computed and appended to this file.</td>
<td>none</td>
</tr>
<tr>
<td>unScaled</td>
<td>Whether to apply checks to the original, unscaled version of the model.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.15.3.2 Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dualCSTol</td>
<td>Tolerance on dual complementary slackness, i.e. between the dual variables and the primal constraints.</td>
<td>1e-7</td>
</tr>
<tr>
<td>dualFeasTol</td>
<td>Tolerance on dual feasibility, i.e. to check feasibility of the dual variables and the dual constraints.</td>
<td>1e-6</td>
</tr>
<tr>
<td>ECTol</td>
<td>Tolerance on equilibrium condition complementarity. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.</td>
<td>1e-6</td>
</tr>
<tr>
<td>primalCSTol</td>
<td>Tolerance on primal complementary slackness, i.e. between the primal variables and the dual constraints.</td>
<td>1e-7</td>
</tr>
<tr>
<td>primalFeasTol</td>
<td>Tolerance on primal feasibility, i.e. to check feasibility of the primal variables and constraints.</td>
<td>1e-6</td>
</tr>
<tr>
<td>showTol</td>
<td>relative tolerance for showSlacks - we only want to see explicit slacks that are relatively small</td>
<td>1e-4</td>
</tr>
</tbody>
</table>

### 5.16 GAMSCHK

Abridged GAMSCHK USER DOCUMENTATION - Version 1.1
A System for Examining the Structure and Solution Properties of GAMS Model Instances

Author

Bruce A. McCarl, Professor, Department of Agricultural Economics, Texas A&M University

Date

April 2013
5.16 GAMSCHK

5.16.1 GAMSCHK USER DOCUMENTATION

This document describes procedures designed to aid users who wish to examine empirical GAMS models for possible flaws. The conceptual basis for many of the routines herein is supplied in McCarl and Spreen, and McCarl et.al.

The function of the specific components of GAMSCHK are to:

- List coefficients for user selected equations and/or variables using the DISPLAYCR procedure.
- List the characteristics of selected groups of variables and/or equations using MATCHIT.
- List the characteristics of equation and variable blocks using BLOCKLIST.
- Examine a GAMS model to see whether any variables and equations contain specification errors using ANALYSIS.
- Generate schematics depicting the characteristics of coefficients by variable and equation blocks using BLOCKPIC.
- Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude using PICTURE.
- Reconstruct the reduced cost of variables and the activity within equations after a model solution using POSTOPT.
- Help resolving problems with unbounded or infeasible models using NONOPT and ADVISORY.

5.16.2 General Notes on Package Usage

GAMSCHK must replace a solver. This is done using a GAMS option statement of the form:

```gams
OPTION LP= GAMSCHK;
```

or

```gams
OPTION NLP=GAMSCHK;
```

or

```gams
OPTION MIP=GAMSCHK;
```

which replaces either the solver of the particular model type with GAMSCHK. In turn, the user will invoke the solver using the statement:

```gams
SOLVE MODELNAME USING LP MINIMIZING OBJNAME;
```

where MODELNAME is the name used in the GAMS MODEL statement; OBJNAME is the objective variable name for the model; and the type of solver that GAMSCHK has replaced which must also be able to solve this type of problem (LP, NLP, MIP, ...) is identified. The following are examples of GAMS sequences which can be added to the GAMS file:

```gams
OPTION NLP=GAMSCHK;
SOLVE TRANSPORT USING NLP MINIMIZING Z;
```

or

```gams
OPTION LP=GAMSCHK;
SOLVE FEED USING LP MINIMIZING COST;
```

or

```gams
OPTION MIP=GAMSCHK;
SOLVE RESOURCE USING MIP MAXIMIZING PROFIT;
```
5.16.2.1 Selecting a Procedure and Providing Input - the *.GCK File

GAMSCHK requires that the user indicate which procedures are to be employed. This is specified through the use of the *.GCK file where the * refers to the filename from the GAMS execution instruction. The general form of that file is:

FIRST PROCEDURE NAME
   ITEM SELECTION INPUT

SECOND PROCEDURE NAME
   ITEM SELECTION INPUT

Spaces and capitalization are ignored in this input. For example, a *.GCK file could look like

DISPLAYCR
  variables
     SELL(*,*,FANCY)
  makeetable
Invariables
     transport(plant2,*,fancy)
Equations
     objT
     notthere
in equations
     resourceq(plant1)
PICTURE

The first procedure name in this case is DISPLAYCR and the following 10 lines indicate the items to be selected. Then, we also request PICTURE. Selection entries are treated using several assumptions. In particular:

1. If the *.GCK file is empty then it is assumed that the BLOCKPIC procedure is selected.
2. Spaces maybe freely used in the GCK input file.
3. Upper, lower, or mixed case input is accepted.
4. GAMSCHK recognizes certain words. These words are listed in Appendix A: Reserved Names and cannot be used as variable or equation names.

5.16.2.2 The *.GCK file: General Notes on Item Selection

Some of the procedures permit selection of variables, equations or functions. Specifically, the DISPLAYCR, PICTURE, POSTOPT, and MATCHIT procedures accept input identifying the variables and equations to be utilized. Also NONOPT accepts limited input controlling its function. General observations about the selection requests are

1. Variables can be chosen by entering the word VARIABLE or VARIABLES possibly with a modifier, followed by variable selection statements.

---

2 Thus, if the GAMS instructions are in the file called MYMODEL, and GAMS is invoked using the DOS command GAMS MYMODEL, then the GCK file would be called MYMODEL.GCK. If GAMS instructions are on the filename with a period in it then the name up to the period will be used, i.e., the GCK file associated with MYMODEL.IT would be MYMODEL.GCK
2. Variables can also be selected using the INEQUATION or INEQUATIONS syntax followed by names of equations. Use of this syntax results in selection of variables with coefficients in the named equations.

3. Equations are selected by entering the keyword, EQUATION or EQUATIONS possibly with a modifier, followed by equation selection statements.

4. Equations can also be selected using the INVARIABLE or INVARIABLES syntax followed by names of variables. Use of this syntax results in selection of equations in which the named variables have coefficients.

5. Certain item selection modifier keywords can be used depending on procedure. The INTERSECT keyword works with procedures DISPLAYCR and POSTOPT. The INEQUATION and INVARIABLE keywords work with procedures DISPLAYCR, PICTURE and POSTOPT. LISTEQUATION and LISTVARIABLE keywords work with the MATCHIT procedure. INSOLUTION, NOTINSOLUTION, BINDING, and NOTBINDING keywords work with POSTOPT. The keywords VERBOSE and IDENTIFY work with NONOPT.

6. If variable or equation names do not follow the keyword, then usually all variables or equations are assumed selected.

When variables or equations are to be selected after an item selection keyword, a number of input conventions apply. These conventions are:

1. If a variable or equation name is entered without any following parentheses, then all cases for that variable or equation are selected.

2. The selection entries identify specific elements from among the sets over which the variables and equations are defined. In specifying these elements one can use various wild card entries as discussed below or an element name. Note GAMS set or subset names cannot be used. Set membership information is not available to the GAMSCHK routines.

3. Wild cards can be used to select items. An "*" will select any item. For example, "B*" will select anything starting with a B. "A?B" will select anything beginning with A, ending with B with one intervening alpha numeric character.

4. When individual elements are specified, you need not enclose them in quotes (".").

5. Quotes must be specified to include set item names with spaces, and special characters. In that case wild cards do not work and all input up to the next quote is simply copied.

6. When the selected item has more dimensions than specified, then all later dimensions are handled as if a wild card were specified. For example, when a variable X is defined with reference to 4 sets in the GAMS instructions, but only 3 parameters are specified in the GAMSCHK input, then the request is handled as if all elements of the 4th are desired.

7. When the selected item has less dimensions in GAMS than in the item selection input, then all additional dimensions are ignored. Thus, when a variable X is defined with reference to 3 sets in GAMS, but 4 parameters are specified in the item selection file, then the 4th specification is ignored.

8. Multiple selection statements can appear on successive lines of the *.GCK file. Output is ordered according to the way items are found in the GAMS file which is determined by the ordering of variables, equations, and set elements in the original GAMS input.

9. Error messages will be generated when an entry cannot be matched to a GAMS element.

10. Examples include
<table>
<thead>
<tr>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(*,CLEVELAND)</td>
<td>which indicates that X will be selected for any element of the first set</td>
</tr>
<tr>
<td></td>
<td>where the element in the second set equals CLEVELAND</td>
</tr>
<tr>
<td>X(SEATTLE)</td>
<td>when X is two dimensional selects all cases where the first set element</td>
</tr>
<tr>
<td></td>
<td>is SEATTLE</td>
</tr>
<tr>
<td>X(SEATTLE,CHICAGO,Z)</td>
<td>when X is two dimensional selects the case where the first set element</td>
</tr>
<tr>
<td></td>
<td>equals SEATTLE, and the second element equals CHICAGO. The third</td>
</tr>
<tr>
<td></td>
<td>is ignored.</td>
</tr>
<tr>
<td>X</td>
<td>all X's will be selected</td>
</tr>
<tr>
<td>X(S*, C.O, Z)</td>
<td>when X is three dimensional selects where all X's with first element</td>
</tr>
<tr>
<td></td>
<td>starting with S, second element beginning with C and ending with O</td>
</tr>
<tr>
<td></td>
<td>and third element Z will be selected.</td>
</tr>
<tr>
<td>*</td>
<td>all variables or equations will be selected</td>
</tr>
<tr>
<td>{empty selection set}</td>
<td>all variables or equations will be selected</td>
</tr>
</tbody>
</table>

5.16.2.3 Procedure Output

In all cases the output generated by the procedure will be written to the * .LST file associated with the GAMS call. Thus, if the file is called MODEL with the * .GCK file (MODEL.GCK), then all output will be on MODEL.LST.

5.16.2.4 Nonlinear Terms

GAMS models examined with GAMSCHK may involve nonlinear terms. In such cases, GAMSCHK uses the value of the nonlinear term sent forth from GAMS which is an accurate marginal, not total value. GAMS develops this value based on the current level value of the variable. This will either be: a) the starting point selected by GAMS, if the model has not been solved, or b) the current solution value, if the model has been solved. The most accurate portrayals of the coefficients will be generated after the model has been solved through a GAMS SOLVE command before invoking GAMSCHK. Some cases may require a solution and/or the specification of a good starting point before using GAMSCHK. Also, nonlinear terms potentially cause misleading coefficients as those values are local marginal, not global, values determined by the current levels of the variables. Nonlinear terms are marked with *** in the DISPLAYCR, POSTOPT, and NONOPT output.

5.16.2.5 Entering Comments in the * .GCK File

The * .GCK file has been programmed so that users can enter comments. These comments can take one of two forms. Comments that begin with a hash mark are copied to the output when the program runs. Comments which begin with a question mark are simply overlooked. Thus, one can temporarily comment GAMSCHK selection statements making them inactive by putting in question marks. If multiple procedures are being run or if some sort of output is decided to screen in the computer output then the hash marks can be entered.

5.16.2.6 Controlling Page Width in the * .GCK File

When running multiple procedures, in particular the pictures with other procedures, it is often desirable to have some procedures run with wide page widths, but the rest with a narrower page width. The GCK file provides the option to narrow the page width using a PW= command. In particular, what one can do is run GAMS with a large page width, i.e. run GAMS BLOCK pw=200, then insert in the GCK file instructions which narrow that page width for selected procedures. Users should note that the page width can never be made any wider than the default page width when running with GAMS. Information in excess of the page width will be ignored. Thus, if the model is run under the default status which has a page width of 75 characters then GAMSCHK will reduce the page width down to the maximum page width allowed. Consequently, the pw= command can only be used to narrow the page width from the default page width, not increase it.
5.16.2.7 Running Multiple Procedures

GAMSCHK can run multiple procedures during one job. This is done by simply stacking the sequence of the commands in the .GCK file.

5.16.3 Use of the Procedures

The following section describes the procedures available in GAMSCHK and their input requirements.

5.16.3.1 DISPLAYCR

Brief Purpose: DISPLAYCR displays all coefficients from the empirical model for a set of user selected equations and variables. All nonzero coefficients under each selected variable or in each selected equation are displayed with the associated variable or equation name and coefficient value. The selection entries may refer to all terms in equations under variables or only those coefficients at the intersection of the selected variables and equations.

Usage Notes: This option mirrors the GAMS LIMCOL and LIMROW options, but allows the user to select the specific items to be displayed. Partial displays within a variable or equation are also allowed using INTERSECT. Use of VARIABLE and EQUATION keywords followed by selection statements allows one to select variables and equations. Use of the INVARIABLE command allows users to select the equations which are associated with a particular variable. For example, if one is having trouble with a particular variable and wants to look at competition in the equations in which it appears, then selecting the variable under the INVARIABLE command will display the complete contents of all the equations in which the selected variables have coefficients. Similarly, the INEQUATION command will display the complete contents of all variables which fall in a particular equation. Nonlinear terms are marked with ***. When the keyword INTERSECT is found then only the coefficients at the intersection of the specified equations and variables are selected. Use of INTERSECT with the INVARIABLE syntax results in the named variables and the equations in which they fall being selected. Similarly, use of INTERSECT with the INEQUATION syntax results in selection of the named equations and the variables which fall in those equations.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: The keyword DISPLAYCR is entered followed by optional lines of item selection input identifying the variables and equations to be displayed. This file can contain the keywords VARIABLE, INVARIABLE, EQUATION, and INEQUATION, with each followed by a specification of the items to be selected using the procedure input specification conventions that were described above. The keyword INTERSECT can also be used. Several special cases are relevant:

- If none of the above keywords are found after DISPLAYCR and another procedure name does not follow, then the input is assumed to identify variables.
- If input is found but the VARIABLE or INEQUATION keyword cannot be found then no variables are assumed selected.
- If the VARIABLE keyword is entered, but is followed by the end of file or an Appendix A reserved word and INEQUATION does not appear, then all variables are assumed selected.
- If the EQUATION or INVARIABLE keyword cannot be found, then no equations are assumed selected.
- If the EQUATION keyword is entered, but is followed by the end of the file or a reserved word and the INVARIABLE command does not occur, then all equations are assumed selected.
• The keyword INVARIABLE is allowed. It should be followed by variable selection statements. In turn, DISPLAYCR selects all equations which have nonzero entries under the INVARIABLE selections.

• The keyword INEQUATION may be used. It should be followed by equation selection statements. In turn, DISPLAYCR selects all variables which have nonzero entries in the INEQUATION selections.

• The keyword INTERSECT causes only coefficients at the intersection of the specified equations and variables to be displayed. This occurs for all specifications in this run of DISPLAYCR. One should use DISPLAYCR again if some intersecting and some non-intersecting displays are desired.

• When INTERSECT appears along with INVARIABLE, the named variable is selected along with all the equations in which it falls. Similarly, when INTERSECT and INEQUATION appear then all the named equations and the variables appearing in them are selected.

5.16.3.2 MATCHIT

Brief Purpose: MATCHIT retrieves the names and characteristics of selected variables and equations. The characteristics reported tell whether the items are nonlinear as well as reporting scaling characteristics and counts of the coefficients. MATCHIT will summarize the items which match a request or list all the items individually.

Usage Notes: The input to MATCHIT can include the keywords VARIABLE and EQUATION along with those keywords with the prefix LIST attached. When the LIST prefix is not used, the procedure summarizes the characteristics of all items which match the item requests counting the number of matching items, the number of those items which are nonlinear, the total coefficients under or in those items, the number of positive, negative, and nonlinear coefficients that fall under or in those items. This does not list the names of the individual items which match. If the LIST prefix is used (entering LISTVARIABLE or LISTEQUATION) then the individual matching items are printed in the order in which they are encountered. For each matching item the information tells whether it is nonlinear, how many total coefficients it has, the count of positive, negative, and nonlinear coefficients falling under it, and the minimum and maximum absolute values of coefficients under it (excluding the objective function coefficient).

Note that when GAMS internal scaling features are employed then by default scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: This file contains the keyword MATCHIT, followed by optional item selection input data. The optional input identifies the variables and equations to be displayed. This input can contain the keywords VARIABLE or LISTVARIABLE followed by a specification of the variables to be selected using the procedure input specification conventions that were described above. This can be followed by the keyword EQUATION or LISTEQUATION and the specified entries.

Several special cases are relevant:

• If the procedure name is not followed by any selection input, then a count of all variables and equations appears.

• If the input is found, but the input does not begin with VARIABLE, EQUATION, LISTVARIABLE, or LISTEQUATION keywords, then the input is assumed to contain variable names.

• If the VARIABLE keyword is entered, but is not followed by variable selection statements, and LISTVARIABLE does not appear, then all variables are assumed selected.

• If the EQUATION or LISTEQUATION keyword cannot be found, then equations are assumed selected.
• If the EQUATION keyword is entered, but is not followed by equation selection statements or a LISTEQUATION entry, then all equations are assumed selected.

• The keyword LISTVARIABLE is allowed. It should be followed by variable selection statements. In turn, MATCHIT lists all variables which fall under the request.

• The keyword LISTEQUATION may also be used. It should be followed by equation selection statements. In turn, MATCHIT lists all equations which fall under the request.

5.16.3.3 ANALYSIS

Brief Purpose: Analyzes the structure of all variables and equations. Information is given on errors involving obvious model misspecifications causing redundancy, zero variable values, infeasibility, unboundedness, or obvious constraint relaxations in linear programs. The checks are those identified in Tables 1, 2 and 3.

Usage Notes: The analysis tests given in Tables 1 and 2 are utilized to determine if individual variables or equations in the model possess obvious specification errors. One test, for example, considers whether or not in a maximization problem a variable appears which has a positive return in the objective function, but no coefficients in the constraints indicating an obviously unbounded model. Similarly, information is provided on whether certain equations can never be satisfied. For example, tests examine whether an equality equation appears with a negative right hand side and all positives on the left hand side. Also tests see whether the bounds on variables preclude equation satisfaction or make equations redundant (Table 3). In ANALYSIS these tests are applied to each and every variable and equation. The BLOCKPIC and BLOCKLIST routines utilize the tests on a block by block basis. Thus, the messages will be triggered only if every variable or equation in that block has the same problem. Also interactions between variables and equations are not checked so ANALYSIS only finds flaws contained in individual variables/equations.

Input File: The keyword ANALYSIS is all that is accepted.

5.16.3.4 BLOCKLIST

Brief Purpose: The BLOCKLIST procedure displays the number and characteristics of the items in each GAMS variable and equation block.

Usage Notes: The characteristic information gives:

1. The variable sign restriction or equation inequality type.
2. The number of variables or equations in this block;
3. The number of variables or equations with at least one nonlinear term in this block.
4. The number of positive coefficients under the variables or in the equations.
5. The number of negative coefficients under the variables or in the equations.
6. The number of nonlinear coefficients under the variables or in the equations.
7. The largest coefficient in absolute value in this block;
8. The smallest coefficient in absolute value in this block. Analysis tests are also performed as discussed under the ANALYSIS procedure.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: No input other than the procedure name is needed.
5.16.3.5 BLOCKPIC

**Brief Purpose:** Generates model schematics and scaling information. The schematics depict coefficient signs, total and average number of coefficients within each GAMS equation and variable block.

**Usage Notes:** These schematics are designed to aid users in identifying flaws in coefficient placement and sign. The summary information on problem scaling characteristics is designed to help users in scaling data. The scaling information is usually reported after any GAMS scaling (using the variablename.scale and equationname.scale features) but before solver scaling. (The user can change whether descaling is done - see the options file). Analysis tests are done using the procedures in Tables 1 and 2.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** The keyword BLOCKPIC is all that is recognized.

5.16.3.6 PICTURE

**Brief Purpose:** Generates a schematic depicting the location, sign and magnitude of coefficients for selected variables and equations. Users can use this schematic to help identify flaws in coefficient placement, magnitude, or sign. Reports are also generated on the number of individual elements in the pictured portions of each variable and equation.

**Usage Notes:** This output can be quite large, so PICTURE should only be used for small models or model components. Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** Optional input instructions may appear after the PICTURE keyword. This input selects the variables and equations to be included. Only coefficients at the intersection of the selected variables and equations are portrayed. The selected item in the .GCK file can contain the keywords VARIABLE, or INvariable followed by a specification of the selected variables using the procedure input specification conventions above. This can be followed by the keywords EQUATION or INEquation and the specified entries. Several special cases are also relevant:

- If the VARIABLE or INEQUATION keywords cannot be found, then all variables are assumed selected.
- If the EQUATION or INVARIABLE keywords cannot be found, then all equations are assumed to selected.
- If the none of the VARIABLE, INVARIABLE, EQUATION, or INEQUATION keywords are found, everything is pictured and all other input is ignored.
- When the INVARIABLE keyword is used, then all equations in which those variables have coefficients are selected along with the named variables.
- When the INEQUATION keyword is used, then all variables which have coefficients in the named equations are selected along with the named equations.
5.16.3.7 POSTOPT

**Brief Purpose:** Does post optimality computations. In that capacity POSTOPT either:

- Reconstructs the reduced cost of variables after a GAMS model solution. Modelers can use this information to discover why certain variables are nonbasic or why certain shadow prices take on particular values, or
- Reconstructs the usage and supply across an equation after a GAMS model solution. Modelers can use this information to discover why certain variables or slacks take on particular values, as well as to find out where items within equations are produced and/or used.

**Usage Notes:** POSTOPT uses essentially the same input conventions as does DISPLAYCR. Thus, the usage notes in that selection are also relevant here. In addition:

1. POSTOPT requires a solution has been obtained GAMSCHK will automatically cause a solver to be invoked unless suppressed by the options file;
2. Nonlinear terms may not be accurate in the row sums as their marginal value not their total value is used but GAMS will have adjusted the right-hand sides for their presence; and
3. Attention can be restricted to only certain types of variables or equations. Variables that are INSOLUTION (Nonzero or with Zero marginals), NOTINSOLUTION (zero with a nonzero marginal) can be requested, BINDING or NONBINDING equations can be focused on.

Note that when GAMS internal scaling features are employed, the default option is that the unscaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** An optional input file is read in, indicating the specific variables desired using the conventions explained under DISPLAYCR above. In addition:

- One can enter INSOLUTION to restrict attention to variables which are nonzero or have zero marginals.
- One can enter NOTINSOLUTION to restrict attention to zero variables.
- The above entries restrict alteration in all VARIABLE or INEQUATION selection statements in a POSTOPT run.
- One can enter BINDING to only consider equations with zero slack. Similarly, NONBINDING considers equations with nonzero slack.
- The above equation specifications restrict all sections by all EQUATION or INVARIABLES items in a POSTOPT run.

5.16.3.8 ADVISORY

**Brief Purpose:** To identify variables which could be unbounded or equations and variable bounds which could cause a model to be infeasible.

**Usage Notes:** The ADVISORY procedure causes a presolution report on the set of all: a) variables which could be unbounded and/or b) equations and variable bounds which could cause infeasibility. The tests used are summarized in Table 3. This procedure identifies all variables which would need to be bounded as well as all constraints which need artificial variables if one wishes to diagnose problems in a model. The same output is also generated by NONOPT but the ADVISORY version does not require a solution.

**Input file:** Just the word ADVISORY
5.16.3.9 NONOPT

**Brief Purpose:** To help diagnose unbounded and infeasible models.

**Usage Notes:** The NONOPT procedure can be used in either an informative mode or with models which terminate as unbounded or infeasible. NONOPT will look through an optimal model reporting all variables which may be potentially unbounded or infeasible and all equations which may be infeasible using the checks explained under the ADVISORY section. Also in an unbounded model NONOPT can report the names of unbounded or infeasible variables or equations as well as either budgeting or row summing them. NONOPT runs after a solution and causes a solve to occur.

**Input File:** NONOPT may be followed by optional keywords IDENTIFY or VERBOSE. The IDENTIFY keyword causes GAMSCHK to report potential unbounded variables and/or infeasible equations. VERBOSE causes full budgets and row summing as done by the POSTOPT procedure on infeasible equations, and/or variables as well as unbounded variables and/or equations. Only the last encountered of the VERBOSE or IDENTIFY keywords will be obeyed. The details on these options are as follows:

1. If the IDENTIFY keyword is used, then the rules in Table 3 are applied to the model. Identify also anticipates that large upper bounds and/or artificial variables may be present. In an optimal condition all variable and equation levels that have exponents greater than the user supplied level filter in the options file (or 6 by default) are identified as items which could be involved with an unbounded model. Similarly, all variables or equations with marginals greater in exponent than the user supplied marginal exponent filter will be identified as items potentially involved with an infeasible model.

2. When the VERBOSE keyword is read then all variables and equations which are listed as nonoptimal or infeasible are treated using the budgeting and row summing aspects of POSTOPT.

3. When no keyword is found and the model solution is not optimal then the nonoptimal equations, infeasible equations and/or nonoptimal variables automatically listed.

5.16.4 Options File

GAMSCHK accepts an option file controlling solver choice (when needed); descaling; and size of the nonoptimal filters; the number of variable and column blocks selection entries allowed. The file is called GAMSCHK.OPT

5.16.4.1 Solver Choice Options

GAMSCHK calls for the solution of the problem when the POSTOPT or NONOPT procedures are used. In doing this, GAMSCHK internally selects the default GAMS solver for a problem class. Users may override this choice using the solver options file. Users may also force or suppress the solution process.

There are 16 solver related keywords allowed in the options file. These are as follows:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>Gives name of solver for LP problems</td>
</tr>
<tr>
<td>MIP</td>
<td>Gives name of solver for MIP problems</td>
</tr>
<tr>
<td>RMIP</td>
<td>Gives name of solver for RMIP problems</td>
</tr>
<tr>
<td>NLP</td>
<td>Gives name of solver for NLP problems</td>
</tr>
<tr>
<td>MCP</td>
<td>Gives name of solver for MCP problems</td>
</tr>
<tr>
<td>MPEC</td>
<td>Gives name of solver for MPEC problems</td>
</tr>
</tbody>
</table>
5.16 GAMSCHK

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMPEC</td>
<td>Gives name of solver for RMPEC problems</td>
</tr>
<tr>
<td>CNS</td>
<td>Gives name of solver for CNS problems</td>
</tr>
<tr>
<td>DNLP</td>
<td>Gives name of solver for DNLP problems</td>
</tr>
<tr>
<td>MINLP</td>
<td>Gives name of solver for MINLP problems</td>
</tr>
<tr>
<td>RMINLP</td>
<td>Gives name of solver for RMINLP problems</td>
</tr>
<tr>
<td>SOLVERNAME</td>
<td>Gives name of solver to be used regardless of problem type</td>
</tr>
<tr>
<td>NOSOLVE</td>
<td>Suppresses solution of the problem</td>
</tr>
<tr>
<td>SOLVE</td>
<td>Forces solution of the problem</td>
</tr>
<tr>
<td>DESCALE</td>
<td>Controls treatment of scaling</td>
</tr>
<tr>
<td>OPTFILE</td>
<td>Solver options file number</td>
</tr>
</tbody>
</table>

In the first five cases, the option name is followed by the name of one of the licensed solvers. If the options file is empty, then the default solver will be used. If a solver name is given, then that solver will be used provided it matches the name of a solver GAMS recognizes.

5.16.4.2 When Should I Use SOLVE or NOSOLVE

Ordinarily GAMSCHK will cause a solver to be used if either the POSTOPT or the NONOPT options are used. However, users can force solutions under other cases or suppress solutions if desired.

One should only force a solution (using the SOLVE option) when one wishes to use the solution information after GAMSCHK is done either to examine the solution output or do post optimality calculations. Forcing a solution will not cause GAMSCHK to have improved representations of nonlinear terms. That will only occur when a SOLVE statement is executed before the SOLVE statement involving GAMSCHK.

5.16.4.3 Control of Number of Variable and Row Selections Allowed

The GAMSCHK program uses an upper estimate on the number of variable or equation blocks. In rare circumstances users may wish to override this choice. The options for this are:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARBLOCK</td>
<td>Maximum number of variable blocks allowed</td>
</tr>
<tr>
<td>EQUBLOCK</td>
<td>Maximum number of equation blocks allowed</td>
</tr>
</tbody>
</table>

These options are followed by a number, but should not be routinely used.

5.16.4.4 Scaling

GAMS users may be utilizing internal features which involve scaling through the Model-name.SCALEOPT=1, VariableName.SCALE, and EquationName.SCALE options. GAMSCHK can work with these options to create output which reflects scaled, unscaled or partially unscaled output. In particular, the command DESCALE can be entered with one of three options: NEVER, ALL, or PART. If you enter NEVER, then none of the model output will be descaled. If you enter ALL, then all of the model output will be descaled. The third option is to use PART. In that case the NONOPT and POSTOPT output will be descaled whereas scaled information will be displayed for PICTURE, BLOCKPIC, BLOCKLIST, MATCHIT and DISPLAYCR. The PART option allows investigation of scaling. If you do not enter a DESCALE option then all information will be reported as if the PART option was chosen.
5.16.4.5 NONOPT Filters

The NONOPT model in "IDENTIFY" mode checks through a model solution to identify large marginals and/or large variable values. The limits on these checks are provided by two options:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVELFILT</td>
<td>Numerical value of exponent on &quot;unbounded levels&quot;</td>
</tr>
<tr>
<td>MARGFILT</td>
<td>Numerical value of exponent on &quot;infeasible marginals&quot;</td>
</tr>
</tbody>
</table>

These options provide upper bounds on the exponents of the absolute values for the levels and marginals. They are followed by an integer which gives the exponent. Thus, entries like

```
LEVELFILT 7
MARGFILT 7
```

will cause the reporting of all marginals and levels which are greater in absolute value than $10^7$.

5.16.4.6 Example Options File

The GAMSCHK option file is called GAMSCHK.OPT. An example of a file could look like the following 6 lines:

```
LP SOPLEX
MIP CBC
VARBLOCK 50
SOLVE
DESCALE PART
LEVELFILT 4
```

5.16.4.7 Solver Options File

One other important aspect regarding the options file involves the use of a problem solver options file when a solver such as SOPLEX, CBC, IPOPT etc. is also being used. As seen above the GAMSCHK.OPT does not recognize option commands such as those which would be submitted to the programming model solvers - SOPLEX for example. In all cases GAMSCHK will cause the default option file for the solver to be used when invoking the solver. Thus if SOPLEX and the options file is invoked is being used, SOPLEX options are controlled by the option file SOPLEX.OPT while GAMSCHK.OPT controls GAMSCHK operation. Users can change the number of the solver options file being used by using the OPTFILE parameter in the options file. OPTFILE 2 would cause use of solver options file .OP2.

5.16.5 Known Bugs

There are a few bugs that can cause GAMSCHK to report improper outputs or results. A list of the known bugs, their symptoms and a remedy is given below.
<table>
<thead>
<tr>
<th>Symptom</th>
<th>Cause</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero Shadow Prices in POSTOPT</td>
<td>Old GAMS version or no Prior Solve</td>
<td>1) Make sure the model was solved, 2) if it was, do not suppress solve in option file, or 3) update to most recent GAMS version</td>
</tr>
<tr>
<td>Descaling Does Not Work</td>
<td>Old Version of GAMS</td>
<td>Update</td>
</tr>
<tr>
<td>GAMS Blows up after GAMSCHK Runs</td>
<td>Old GAMS version</td>
<td>Ignore, *.LST file, results are fine, can be fixed by updating to the most recent version of GAMS</td>
</tr>
<tr>
<td>POSTOPT has error in budgets equal to twice objective function coefficient for nonlinear maximizations</td>
<td>Old GAMS MINOS version</td>
<td>Switch to a minimization formulation or update GAMS/MINOS</td>
</tr>
<tr>
<td>ROWSUM does not fully account for the value of nonlinear terms in POSTOPT</td>
<td>Value of nonlinear terms sent from GAMS are only a marginal value</td>
<td>None planned. GAMSCHK would need reprogramming</td>
</tr>
<tr>
<td>Error message about size of VB/BLOCK or EBN/BLOCK</td>
<td>exceeded maximum number of blocks</td>
<td>Modify option file, enlarging or eliminating parameters</td>
</tr>
<tr>
<td>GAMSCHK won’t run</td>
<td>Files are not properly installed</td>
<td>Recheck installation. If still doesn’t work report to author</td>
</tr>
<tr>
<td>Zero shadow prices when using NOSOLVE</td>
<td>Old version of GAMS solvers or Shadow prices suppressed</td>
<td>Try changing GAMSCOMP.TXT lines 2 or 0 to 12 or 10, if that doesn’t work update GAMS.</td>
</tr>
</tbody>
</table>

5.16.6 Tables

Table 1: Conditions under which a modeler should be advised of potential difficulty for equations without nonlinear terms.

- **a/** The PS cases indicate, because the variables in this equation follow this pattern, that:
  1. The variables appearing with nonzeros in this equation are forced to equal zero.
  2. This equation can never be satisfied and is obviously infeasible.
  3. This equation is redundant. The nonnegativity conditions are a stronger restriction.
- **b/** In the examples x denotes indexed non-negative variables, y indexed non-positive variables, and z a single unrestricted variable.
- **c/** Here and in the cases below at least one nonzero must occur.
- **d/** These entries give examples of the problem covered by each warning. Namely, in the first case examining only the nonnegative variables suppose all those variables have signs ≥ 0 but the right-hand-side is zero. Thus, we have X ≥ 0 and X ≤ 0 which implies X = 0. A warning is generated in that case.
- **e/** Only one coefficient is allowed.
<table>
<thead>
<tr>
<th>Type of constraint</th>
<th>Count of coefficients under a variable of this type with a particular sign</th>
<th>Sign of RHS</th>
<th>Type of PS⁹/</th>
<th>Examples⁹/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nonnegative</td>
<td>Nonpositive</td>
<td>Unrestricted</td>
<td></td>
</tr>
<tr>
<td>≤</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>≤</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>≤</td>
<td>0</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>=</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>=</td>
<td>0</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>≥</td>
<td>0</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>≥</td>
<td>0</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>≥</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>≥</td>
<td>0</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>≥</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>≥</td>
<td>≥ 0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
</tr>
</tbody>
</table>
Table 2: Conditions under which a modeler should be warned about variables in a maximization problem.

<table>
<thead>
<tr>
<th>Type of Variable</th>
<th>Objective function coefficient sign</th>
<th>Number of $a_{ij}$’s of a sign in $\geq$ rows</th>
<th>$=$ rows</th>
<th>$\leq$ rows</th>
<th>PS$^a$/ Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonnegative</td>
<td>$+$</td>
<td>$\geq0$ 0 0 0 0 0 $\geq0$</td>
<td></td>
<td></td>
<td>Unbounded Variable case 1 $\max x^b/ x + DQ \geq -x + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>$-$</td>
<td>0 $\geq0$ 0 0 $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Zero optimal solution case 2 $\max -z/ x + DQ \geq x + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq0$ 0 0 0 $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Variable Relaxes constraint case 3 $\max ox/ x + DQ \geq -x + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq0$ 0 $\geq0\geq0$ $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Variable Relaxes constraint case 4 $\max ox/ x + DQ \geq x + EQ \geq 0$</td>
</tr>
<tr>
<td>Nonpositive</td>
<td>$-$</td>
<td>0 $\geq0$ 0 0 $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Unbounded Variable case 1 $\max -y/ y + DQ \geq y + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>$+$</td>
<td>$\geq0$ 0 0 0 0 $\geq0$</td>
<td></td>
<td></td>
<td>Zero optimal solution case 2 $\max y^b/ y + DQ \geq -y + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0 $\geq0$ 0 0 $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Variable Relaxes constraint case 3 $\max ox/ -y + DQ \geq y + EQ \geq 0$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq0$ 0 $\geq0\geq0$ $\geq0$ 0 $\geq0$</td>
<td></td>
<td></td>
<td>Variable Relaxes constraint case 4 $\max ox/ -y + DQ \geq y + EQ \geq 0$</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>+/-</td>
<td>0 0 0 0 0 0 $\geq0$</td>
<td></td>
<td></td>
<td>Unbounded Variable case 1 $\max \pm z$</td>
</tr>
</tbody>
</table>

- **a/** PS cases are: The variables which satisfy this condition are:
  1. Unbounded as they contribute to the objective function while satisfying the constraints.
  2. Obviously zero since they consume constraint resources and have a cost in the objective function.
  3. Warning this variable relaxes all constraints in which it appears
  4. Warning this variable relaxes all the equality constraints in which it appears in one direction

- **b/** Here $x(y)$ has a positive objective term and can be increased without ever violating any constraints so $x(y)$ is unbounded.

- **c/** Only one coefficient can be present in the equality rows
Table 3: Conditions When Model Elements Could be Unbounded or Infeasible.

Conditions for Potential Unbounded Variables – Presence of Bounds

<table>
<thead>
<tr>
<th>Types of Variables</th>
<th>Sign of Objective in Max Problem</th>
<th>Upper</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0^a/</td>
<td>+</td>
<td>None</td>
<td>-^b/</td>
</tr>
<tr>
<td>≤0</td>
<td>-</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>+</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>-</td>
<td>-</td>
<td>None</td>
</tr>
</tbody>
</table>

- **a/** If a non negative variable has a positive objective function coefficient without an upper bound, then the variable could be unbounded.
- **b/** Any reasonable value can exist for this item

Conditions for Potential Infeasibility Caused by Bounds on Variables

<table>
<thead>
<tr>
<th>Types of Variables</th>
<th>Existence of Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0^c/</td>
<td>+</td>
</tr>
<tr>
<td>≤0</td>
<td>-</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>+</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>-</td>
</tr>
</tbody>
</table>

- **c/** If a nonnegative variable has a positive lower bound then it could cause infeasibility.

Conditions for Potential Infeasibility in Equations

<table>
<thead>
<tr>
<th>Type of Equations</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤^d/</td>
<td>-</td>
</tr>
<tr>
<td>≥</td>
<td>+</td>
</tr>
<tr>
<td>=</td>
<td>+ or -</td>
</tr>
</tbody>
</table>

- **d/** When a less than or equal equation is present it may not be able to be satisfied if it has a negative RHS.
Table 4: Conditions for Potential Infeasibility or Redundancy in Equations Based on Bounds on Variables.

<table>
<thead>
<tr>
<th>TYPE OF CONSTRAINT</th>
<th>PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq b )</td>
<td>( \geq b )</td>
</tr>
<tr>
<td>SUM OF THE SMALLEST VALUE(^a/)</td>
<td>&gt;b</td>
</tr>
<tr>
<td>SUM OF THE LARGEST VALUE(^b/)</td>
<td>&lt;b</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:

- **a/** Suppose \( X_j \) is bounded with \( LB_j \) (lower bound) \( \leq X_j \leq UB_j \) (upper bound), and we have the sum evaluated at the lower bounds will be the smallest value which could happen in that sum. If the constraint is \( < b \), then if the sum is \( > b \), we know that this constraint will never be satisfied. If the constraint is \( > b \), and the sum is \( > b \), we know that this constraint will not limit any possible \( X \) value. Hence, it is redundant.

- **b/** Suppose \( X_j \) is bounded as follows, \( LB_j \) (lower bound) \( \leq X_j \leq UB_j \) (upper bound), and we have the sum evaluated at the upper bounds which is either \( > b \) or \( < b \), in that sum. If the sum is \( < b \), and the constraint holds it \( < b \) then we know that this constraint will not limit any possible \( X \) value. Hence, it is redundant. If the constraint holds it greater than \( b \), but the sum is \( < b \), we know that this constraint will never be satisfied.

- **c/** Thanks to Paul Preckel for bringing these tests to the authors’ attention.
5.16.7 Appendix A: Reserved Names

VARIABLE
VARIABLES
EQUATION
EQUATIONS
IN_VARIABLE
IN_VARIABLES
INEQUATION
INEQUATIONS
LIST_VARIABLE
LIST_VARIABLES
LIST_EQUATION
LIST_EQUATIONS
POSTOPT
DISPLAYCR
PICTURE
BLOCKPIC
ANALYSIS
MATCHIT
BLOCKLIST
NONOPT
INSOLUTION
NOT_INSOLUTION
NONINSOLUTION
VERBOSE
ADVISORY
BINDING
NONBINDING
INTERSECT
IDENTIFY
PW=
5.16.8 Appendix B: GAMSCHK One Page Summary

Invoking GAMSCHK

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Allowed SubKEYWORDS</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAYCR</td>
<td>VARIABLE*</td>
<td>Displays coefficients of selected variables and equations</td>
</tr>
<tr>
<td></td>
<td>INvariable*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equations are wanted in which selected variables fall</td>
</tr>
<tr>
<td></td>
<td>INEQUATION*</td>
<td>Indicates variables are wanted that fall in selected equations</td>
</tr>
<tr>
<td></td>
<td>INTERSECT++</td>
<td>Show coefficients which appear at intersections of selected var/eqn</td>
</tr>
<tr>
<td>MATCHIT</td>
<td>VARIABLE*</td>
<td>List variable and equation names and summarize characteristics</td>
</tr>
<tr>
<td></td>
<td>LISTVARIABLE*</td>
<td>Summarizes all variables matching selection statements</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Summarizes all equations matching a selection statement</td>
</tr>
<tr>
<td></td>
<td>LISTEQUATION*</td>
<td>Lists each equation matching a selection statement</td>
</tr>
<tr>
<td>ANALYSIS</td>
<td></td>
<td>Checks for obvious structural defects</td>
</tr>
<tr>
<td>BLOCKLIST</td>
<td></td>
<td>Summarizes characteristics of variable and equation blocks</td>
</tr>
<tr>
<td>BLOCKPIC</td>
<td></td>
<td>Generates block level schematics</td>
</tr>
<tr>
<td>PICTURE</td>
<td>VARIABLE*</td>
<td>Generates tableau schematics</td>
</tr>
<tr>
<td></td>
<td>INvariable*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equations are wanted in which selected variables fall</td>
</tr>
<tr>
<td></td>
<td>INEQUATION*</td>
<td>Indicates equation selections follow</td>
</tr>
<tr>
<td></td>
<td>INTERSECT++</td>
<td>Indicates variables are wanted that fall in selected equations</td>
</tr>
<tr>
<td>POSTOPT</td>
<td>VARIABLE*</td>
<td>Reconstructs reduced cost and equation activity</td>
</tr>
<tr>
<td></td>
<td>INvariable*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equation selections follow</td>
</tr>
<tr>
<td></td>
<td>INEQUATION*</td>
<td>Indicates variables are wanted that fall in selected equations</td>
</tr>
<tr>
<td></td>
<td>INTERSECT++</td>
<td>Show coefficients which appear at intersections of selected var/eqn</td>
</tr>
<tr>
<td></td>
<td>NOTINSOLUTION++</td>
<td>Only nonzero vars or those with zero reduced cost</td>
</tr>
<tr>
<td></td>
<td>INSOLUTION++</td>
<td>Only zero vars will be selected</td>
</tr>
<tr>
<td></td>
<td>BINDING++</td>
<td>Only eqns with zero slack will be computed</td>
</tr>
<tr>
<td></td>
<td>NONBINDING++</td>
<td>Only eqns with nonzero slack will be computed</td>
</tr>
<tr>
<td>ADVISORY</td>
<td></td>
<td>List potential infeasible and unbounded items</td>
</tr>
<tr>
<td>NONOPT</td>
<td>IDENTIFY</td>
<td>Lists potential or actual nonoptimal items</td>
</tr>
<tr>
<td></td>
<td>VERBose</td>
<td>Same as ADVISORY but after solution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Does POSTOPT computations on nonoptimals</td>
</tr>
</tbody>
</table>

Other Notes

- Items marked above with an * are followed by item selection statements.
- Items marked with ++ modify the types of variables, equations and coefficients selected.
- In item selection an * is a wild card for multiple characters while a . is a wildcard for one character.
- Spaces and capitalization don't matter in any of the input.
- Options file controls scaling, solver choice, nonopt filters and maximum allowed selections.
- Page width is controlled by a PW= keyword but cannot exceed GAMS page width.
- Lines beginning with a ? or a # are treated as comments.
5.16.9 Appendix C: Summary of GAMSCHK Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS</td>
<td>solver for CNS problems</td>
<td></td>
</tr>
<tr>
<td>DESCALE</td>
<td>controls treatment of scaling</td>
<td>part</td>
</tr>
<tr>
<td>DNLP</td>
<td>solver for DNLP problems</td>
<td></td>
</tr>
<tr>
<td>EQUBLOCK</td>
<td>maximum number of equation blocks allowed</td>
<td>-5</td>
</tr>
<tr>
<td>LEVELFILT</td>
<td>numerical value of exponent on &quot;unbounded levels&quot;</td>
<td>6</td>
</tr>
<tr>
<td>LP</td>
<td>solver for LP problems</td>
<td></td>
</tr>
<tr>
<td>MARGFILT</td>
<td>numerical value of exponent on &quot;infeasible marginals&quot;</td>
<td>6</td>
</tr>
<tr>
<td>MCP</td>
<td>solver for MCP problems</td>
<td></td>
</tr>
<tr>
<td>MINLP</td>
<td>solver for MINLP problems</td>
<td></td>
</tr>
<tr>
<td>MIP</td>
<td>solver for MIP problems</td>
<td></td>
</tr>
<tr>
<td>MPEC</td>
<td>solver for MPEC problems</td>
<td></td>
</tr>
<tr>
<td>NLP</td>
<td>solver for NLP problems</td>
<td></td>
</tr>
<tr>
<td>NOSOLVE</td>
<td>suppresses solution of the problem</td>
<td></td>
</tr>
<tr>
<td>OPTFILE</td>
<td>solver options file number</td>
<td></td>
</tr>
<tr>
<td>RMINLP</td>
<td>solver for RMINLP problems</td>
<td></td>
</tr>
<tr>
<td>RMIP</td>
<td>solver for RMIP problems</td>
<td></td>
</tr>
<tr>
<td>RMPEC</td>
<td>solver for RMPEC problems</td>
<td></td>
</tr>
<tr>
<td>SOLVE</td>
<td>forces solution of the problem</td>
<td></td>
</tr>
<tr>
<td>SOLVERNAME</td>
<td>solver for any problems</td>
<td></td>
</tr>
<tr>
<td>VARBLOCK</td>
<td>maximum number of variable blocks allowed</td>
<td>-5</td>
</tr>
</tbody>
</table>

5.16.10 GAMSCHK References

- McCarl, B.A. GAMSCHK. Older version of GAMSCHK User Documentation with additional GAMS examples.
5.17  Gurobi 9

Gurobi Optimization, www.gurobi.com

5.17.1  Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP) and mixed-integer quadratically constrained programming (MIQCP) solvers.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

While numerous solving options are available, Gurobi automatically calculates and sets most options at the best values for specific problems. All Gurobi options available through GAMS/Gurobi are summarized at the end of this chapter.

We offer a GAMS/Gurobi-Link license that works in combination with a Gurobi callable library license from Gurobi Optimization Inc.

Finally, a bare-bone interface to the LP and MIP solver of Gurobi is available under the name OSIGUROBI. Note that OSIGUROBI lacks several features of the GAMS/Gurobi-Link. It comes free of charge with any GAMS system.

5.17.2  How to Run a Model with Gurobi

The following statement can be used inside your GAMS program to specify using Gurobi

```
Option LP = Gurobi; { or MIP or RMIP or QCP or MIQCP or RMIQCP }
```

The above statement should appear before the `solve` statement. If Gurobi was specified as the default solver during GAMS installation, the above statement is not necessary.

5.17.3  Overview of GAMS/Gurobi

5.17.3.1  Linear, Quadratic and Quadratic Constrained Programming

Gurobi can solve LP and convex QP problems using several alternative algorithms, while the only choice for solving convex QCP is the parallel barrier algorithm. The majority of LP problems solve best using Gurobi’s state-of-the-art dual simplex algorithm, while most convex QP problems solve best using the parallel barrier algorithm. Certain types of LP problems benefit from using the parallel barrier or the primal simplex algorithms, while for some types of QP, the dual or primal simplex algorithm can be a better choice. If you are solving LP problems on a multi-core system, you should also consider using the concurrent optimizer. It runs different optimization algorithms on different cores, and returns when the first one finishes.
GAMS/Gurobi also provides access to the Gurobi infeasibility finder. The infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Gurobi reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. The infeasibility finder is activated by the option IIS. Another option for analyzing infeasible model the FeasOpt option which instructs GAMS/Gurobi to find a minimal feasible relaxation of an infeasible model. See section Feasible Relaxation for details.

GAMS/Gurobi supports sensitivity analysis (post-optimality analysis) for linear programs which allows one to find out more about an optimal solution for a problem. In particular, objective ranging and constraint ranging give information about how much an objective coefficient or a right-hand-side and variable bounds can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or the bounds and right-hand side. GAMS/Gurobi reports the sensitivity information as part of the normal solution listing. Sensitivity analysis is activated by the option Sensitivity.

The Gurobi presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Gurobi can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option ReRun. In default mode only problems that are small (i.e. demo sized) will be rerun.

Gurobi can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Gurobi will automatically use an advanced basis or solution from a previous solve statement. The GAMS BRatio option can be used to specify when not to use an advanced basis/solution. The GAMS/Gurobi option UseBasis can be used to ignore or force a basis/solution passed on by GAMS (it overrides BRatio). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS BRatio option to 1.

5.17.3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear or quadratic programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with discrete variables, Gurobi uses a branch and cut algorithm which solves a series of subproblems, LP subproblems for MILP, QP subproblems for MIQP, and QCP subproblems or LP outer approximation subproblems for MIQCP. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory. With option nonConvex Gurobi can also solve nonconvex (MI)QP and (MI)QCP problems using a spatial branch-and-bound method.

GAMS/Gurobi supports Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution.

If you specify some or all values for the discrete variables together with GAMS/Gurobi option MipStart, Gurobi will check the validity of the values as an integer-feasible solution. If this process succeeds, the solution will be treated as an integer solution of the current problem.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.
5.17.3.3 Nonlinear Programming

Gurobi is not a general purpose nonlinear programming solver, but it is able to handle certain nonlinear constraints by reformulating them into supported linear and/or quadratic constraints. These constraints are:

- **MAX constraint:**
  
  \[ \text{eq1. } r = \max(x_1, x_2, x_3, \ldots, c); \]
  \[ \text{eq2. } r = \text{smax}(i, x(i)); \]

- **MIN constraint:**
  
  \[ \text{eq1. } r = \min(x_1, x_2, x_3, \ldots, c); \]
  \[ \text{eq2. } r = \text{smin}(i, x(i)); \]

- **AND constraint:**
  
  \[ \text{eq1. } r = \text{b1 and b2 and b3 and \ldots}; \]
  \[ \text{eq2. } r = \text{sand}(i, b(i)); \]

- **OR constraint:**
  
  \[ \text{eq1. } r = \text{b1 or b2 or b3 or \ldots}; \]
  \[ \text{eq2. } r = \text{sor}(i, b(i)); \]

- **ABS constraint:**
  
  \[ \text{eq. } r = \text{abs}(x); \]

- **EXP constraint:**
  
  \[ \text{eq1. } r = \exp(x); \]
  \[ \text{eq2. } r = x^a; \]
  \[ \text{eq3. } r = a^x; \]

  Here, \( a > 0 \) and for eq3 the lower bound of \( x \) must be nonnegative.

- **LOG constraint:**
  
  \[ \text{eq1. } r = \log(x); \]
  \[ \text{eq2. } r = \log_2(x); \]
  \[ \text{eq3. } r = \log_{10}(x); \]

- **SIN / COS / TAN constraint:**
  
  \[ \text{eq1. } r = \sin(x); \]
  \[ \text{eq2. } r = \cos(x); \]
  \[ \text{eq3. } r = \tan(x); \]

- **NORM constraint:**
  
  \[ \text{eq. } r = \text{edist}(x_1, x_2, x_3, \ldots); \]

  Note that a 2-norm constraint can lead to a non-convex quadratic model which is much harder to solve than a convex quadratic or linear model.

- **POLY constraint:**
  
  \[ \text{eq. } r = \text{poly}(x, a_0, a_1, a_2, \ldots); \]
Nonlinear constraints must perfectly match one of the above forms. Otherwise, GAMS/Gurobi will raise a capability error. For example, it is not possible to interchange left-hand-side and right-hand-side of the above constraints.

GAMS/Gurobi offers a second approach to specify nonlinear constraints via linear constraints combined with .genconstrtype. These further support 0-NORM, 1-NORM and INF-NORM constraints.

### 5.17.3.4 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Gurobi is known as FeasOpt (for Feasible Optimization) and turned on by parameter FeasOpt in a GAMS/Gurobi option file.

With the FeasOpt option GAMS/Gurobi accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the .feaspref option. The input value denotes the users willingness to relax a constraint or bound. The more likely it will be that a given bound or constraint will be relaxed. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed. It is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a GAMS/Gurobi solver option file using dot options. The syntax is:

```
(variable or equation).feaspref(value)
```

For example, suppose we have a GAMS declaration:

```plaintext
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the gurobi.opt file can be specified by:

```
feasopt 1
v.feaspref   1
v.feaspref('i1',*) 2
v.feaspref('i1','j2') 0
e.feaspref(*,'j1') 0
e.feaspref('i5','j4') 2
```
First we turn the feasible relaxation on. Furthermore, we specify that all variables $v(i,j)$ have preference of 1, except variables over set element $i1$, which have a preference of 2. The variable over set element $i1$ and $j2$ has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter `FeasOptMode` allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter `FeasOptMode` indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations).

Please check description of parameter `FeasOptMode` for details. Also check example models `feasopt*` in the GAMS Model library.

### 5.17.3.5 Parameter Tuning Tool

The Gurobi Optimizer provides a wide variety of parameters that allow you to control the operation of the optimization engines. The level of control varies from extremely coarse-grained (e.g., the `Method` parameter, which allows you to choose the algorithm used to solve continuous models) to very fine-grained (e.g., the `MarkowitzTol` parameter, which allows you to adjust the precise tolerances used during simplex basis factorization). While these parameters provide a tremendous amount of user control, the immense space of possible options can present a significant challenge when you are searching for parameter settings that improve performance on a particular model. The purpose of the Gurobi tuning tool is to automate this search.

The Gurobi tuning tool performs multiple solves on your model, choosing different parameter settings for each, in a search for settings that improve runtime. The longer you let it run, the more likely it is to find a significant improvement.

A number of tuning-related parameters allow you to control the operation of the tuning tool. The most important is probably `TuneTimeLimit`, which controls the amount of time spent searching for an improving parameter set. Other parameters include `TuneTrials` (which attempts to limit the impact of randomness on the result), `TuneResults` (which limits the number of results that are returned), and `TuneOutput` (which controls the amount of output produced by the tool).

While parameter settings can have a big performance effect for many models, they aren’t going to solve every performance issue. One reason is simply that there are many models for which even the best possible choice of parameter settings won’t produce an acceptable result. Some models are simply too large and/or difficult to solve, while others may have numerical issues that can’t be fixed with parameter changes.

Another limitation of automated tuning is that performance on a model can experience significant variations due to random effects (particularly for MIP models). This is the nature of search. The Gurobi algorithms often have to choose from among multiple, equally appealing alternatives. Seemingly innocuous changes to the model (such as changing the order of the constraint or variables), or subtle changes to the algorithm (such as modifying the random number seed) can lead to different choices. Often times, breaking a single tie in a different way can lead to an entirely different search. We’ve seen cases where subtle changes in the search produce 100X performance swings. While the tuning tool tries to limit the impact of these effects, the final result will typically still be heavily influenced by such issues.

The bottom line is that automated performance tuning is meant to give suggestions for parameters that could produce consistent, reliable improvements on your models. It is not meant to be a replacement for efficient modeling or careful performance testing.
5.17.3.6 Compute Server

The Gurobi Compute Server allows you to use one or more servers to offload all of your Gurobi computations.

Gurobi compute servers support queuing and load balancing. You can set a limit on the number of simultaneous jobs each compute server will run. When this limit has been reached, subsequent jobs will be queued. If you have multiple compute servers, the current job load is automatically balanced among the available servers. By default, the Gurobi job queue is serviced in a First-In, First-Out (FIFO) fashion. However, jobs can be given different priorities. Jobs with higher priorities are then selected from the queue before jobs with lower priorities.

Gurobi Compute Server licenses and software are not included in GAMS/Gurobi. Contact Gurobi directly to inquire about the software and license.

5.17.3.7 Distributed Parallel Algorithms

Gurobi Optimizer implements a number of distributed algorithms that allow you to use multiple machines to solve a problem faster. Available distributed algorithms are:

- A **distributed MIP solver**, which allows you to divide the work of solving a single MIP model among multiple machines. A manager machine passes problem data to a set of worker machines in order to coordinate the overall solution process.

- A **distributed concurrent solver**, which allows you to use multiple machines to solve an LP or MIP model. Unlike the distributed MIP solver, the concurrent solver doesn't divide the work associated with solving the problem among the machines. Instead, each machine uses a different strategy to solve the whole problem, with the hope that one strategy will be particularly effective and will finish much earlier than the others. For some problems, this concurrent approach can be more effective than attempting to divide up the work.

- **Distributed parameter tuning**, which automatically searches for parameter settings that improve performance on your optimization model. Tuning solves your model with a variety of parameter settings, measuring the performance obtained by each set, and then uses the results to identify the settings that produce the best overall performance. The distributed version of tuning performs these trials on multiple machines, which makes the overall tuning process run much faster.

These distributed parallel algorithms are designed to be almost entirely transparent to the user. The user simply modifies a few parameters, and the work of distributing the computation to multiple machines is handled behind the scenes by Gurobi.

5.17.3.7.1 Specifying the Worker Pool  Once you’ve set up a set of one or more distributed workers, you should list at least one of their names in the WorkerPool parameter. You can provide either machine names or IP addresses, and they should be comma-separated.

You can provide the worker access password through the WorkerPassword parameter. All servers in the worker pool must have the same access password.
5.17.3.7.2 Requesting Distributed Algorithms Once you've set up the worker pool through the appropriate parameters, the last step to use a distributed algorithm is to set the `TuneJobs`, `ConcurrentJobs`, or `DistributedMIPJobs` parameter. These parameters are used to indicate how many distinct tuning, concurrent, or distributed MIP jobs should be started on the available workers.

If some of the workers in your worker pool are running at capacity when you launch a distributed algorithm, the algorithm won’t create queued jobs. Instead, it will launch as many jobs as it can (up to the requested value), and it will run with these jobs.

These distributed algorithms have been designed to be nearly indistinguishable from the single machine versions. Our hope is that, if you know how to use the single machine version, you’ll find it straightforward to use the distributed version. The distributed algorithms respect all of the usual parameters. For distributed MIP, you can adjust strategies, adjust tolerances, set limits, etc. For concurrent MIP, you can allow Gurobi to choose the settings for each machine automatically or specify a set of options. For distributed tuning, you can use the usual tuning parameters, including `TuneTimeLimit`, `TuneTrails`, and `TuneOutput`.

There are a few things to be aware of when using distributed algorithms, though. One relates to relative machine performance. Distributed algorithms work best if all of the workers give very similar performance. For example, if one machine in your worker pool were much slower than the others in a distributed tuning run, any parameter sets tested on the slower machine would appear to be less effective than if they were run on a faster machine. Similar considerations apply for distributed MIP and distributed concurrent. We strongly recommend that you use machines with very similar performance. Note that if your machines have similarly performing cores but different numbers of cores, we suggest that you use the `Threads` parameter to make sure that all machines use the same number of cores.

Logging for distributed MIP is very similar to the standard MIP logging. The main differences are in the progress section. The header for the standard MIP logging looks like this:

```
Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
```

By contrast, the distributed MIP header looks like this:

```
Nodes | Utilization | Objective Bounds | Work
Expl Unexpl | Active Sync Comm | Incumbent BestBd Gap | It/Node Time
```

You’ll note that columns three through five show different information. In the distributed MIP log, these columns give information about the utilization of the distributed workers, expressed as percentages. The first of these columns shows the fraction of the preceding time period (the time since the previous progress log line) that the workers spent actively processing MIP nodes. The second column shows the time the workers spent waiting for other workers to complete tasks assigned to them. The final column shows the time spent communicating data between machines. The numbers sum to 100%.

Here is an example of a distributed MIP progress log:

```
Nodes | Utilization | Objective Bounds | Work
Expl Unexpl | Active Sync Comm | Incumbent BestBd Gap | It/Node Time
* 0 - 157344.61033 - - - 0s
* 0 - 41359.940833 - - - 0s
* 0 - 25050.545455 - - - 0s
* 0 - 18150.083886 - - - 0s
* 0 - 14372.871258 - - - 0s
* 0 - 13725.475382 - - - 0s
```
One thing you may find in the progress section is that node counts may not increase monotonically. In this example, a solution found at node 261 is reported before a solution found at node 0. Distributed MIP tries to create a single, unified view of node numbers, but with multiple machines processing nodes independently, possibly at different rates, some inconsistencies are inevitable.

Another difference is the line that indicates that the distributed ramp-up phase is complete. At this point, the distributed strategy transitions from a concurrent approach to a distributed approach. The log line indicates which worker was the winner in the concurrent approach. Distributed MIP continues by dividing the partially explored MIP search tree from this worker among all of the workers.

Another difference in the distributed log is in the summary section. The distributed MIP log includes a breakdown of how runtime was spent:

Runtime breakdown:
  Active: 13.73s (75%)
  Sync: 1.16s (6%)
  Comm: 3.45s (19%)

This is an aggregated view of the utilization data that is displayed in the progress log lines. In this example, the workers spent 75% of runtime actively working on MIP nodes, 6% waiting to synchronize with other workers, and 19% communicating data between machines.

The installation instructions for the Gurobi Remote Services can be found on Gurobi’s web page www.gurobi.com.

5.17.3.7.3 Gurobi Instant Cloud An alternative to setting up your own pool of machines is to use the Gurobi Instant Cloud. You only need a GAMS/Gurobi link license when you solve your problems in the Gurobi Instant Cloud. The cost for the Gurobi license is paid on a per use basis directly to Gurobi. If you follow through the steps on the Gurobi web site, you eventually get the names of the machines Gurobi has started for you in the cloud. In order to use these machines from GAMS/Gurobi, you need to provide a Gurobi license with access instructions for the Gurobi Instant Cloud (detailed instructions for configuring the client license file).
5.17.3.8 Solution Pool

While the default goal of the Gurobi Optimizer is to find one proven optimal solution to your model, with a possible side-effect of finding other solutions along the way, the solver provides a number of parameters that allow you to change this behavior.

By default, the Gurobi MIP solver will try to find one proven optimal solution to your model. It will typically find multiple sub-optimal solutions along the way, which can be retrieved later. However, these solutions aren't produced in a systematic way. The set of solutions that are found depends on the exact path the solver takes through the MIP search. You could solve a MIP model once, obtaining a set of interesting sub-optimal solutions, and then solve the same problem again with different parameter settings, and find only the optimal solution.

To store (some of the) solutions found along the way, you can enable the Solution Pool feature by setting option solnpool. If you'd like more control over how solutions are found and retained, the Gurobi Optimizer has a number of parameters available for this. The first and simplest is PoolSolutions, which controls the size of the solution pool. Changing this parameter won't affect the number of solutions that are found - it simply determines how many of those are retained.

You can use the PoolSearchMode parameter to control the approach used to find solutions. In its default setting (0), the MIP search simply aims to find one optimal solution. Setting the parameter to 1 causes the MIP search to expend additional effort to find more solutions, but in a non-systematic way. You will get more solutions, but not necessarily the best solutions. Setting the parameter to 2 causes the MIP to do a systematic search for the n best solutions. For both non-default settings, the PoolSolutions parameter sets the target for the number of solutions to find.

If you are only interested in solutions that are within a certain gap of the best solution found, you can set the PoolGap parameter. Solutions that are not within the specified gap are discarded.

Obtaining an OPTIMAL optimization return status when using PoolSearchMode=2 indicates that the MIP solver succeeded in finding the desired number of best solutions, or it proved that the model doesn't have that many distinct feasible solutions. If the solver terminated early (e.g., due to a time limit), you PoolObjBound attribute (printed to the log) to evaluate the quality of the solutions that were found. This attribute gives a bound on the objective of any solution that isn't already in the solution pool. The difference between this attribute and ObjBound is that the latter gives a bound on the objective for any solution, and which is often looser than PoolObjBound. The PoolObjBound attribute gives a bound on the objective of undiscovered solutions. Further tree exploration won't find better solutions. You can use this bound to get a count of how many of the n best solutions you found: any solutions whose objective values are at least as good as PoolObjBound are among the n best.

5.17.3.8.1 Solution Pool Example

Let's continue with a few examples of how these parameters would be used. Imagine that you are solving a MIP model with an optimal (minimization) objective of 100. Further imagine that, using default settings, the MIP solver finds four solutions to this model with objectives 100, 110, 120, and 130.

If you set the PoolSolutions parameter to 3 and solve the model again, the MIP solver would discard the worst solution and return with 3 solutions in the solution pool. If you instead set the PoolGap parameter to value 0.2, the MIP solver would discard any solutions whose objective value is worse than 120 (which would also leave 3 solutions in the solution pool).

If you set the PoolSearchMode parameter to 2 and the PoolSolutions parameter to 10, the MIP solver would attempt to find the 10 best solutions to the model. An OPTIMAL return status would indicate that either (i) it found the 10 best solutions, or (ii) it found all feasible solutions to the model, and there were fewer than 10. If you also set the PoolGap parameter to a value of 0.1, the MIP solver would try to find 10 solutions with objective no worse than 110. While this may appear equivalent to asking for 10 solutions and simply ignoring those with objective worse than 110, the solve will typically complete significantly faster with this parameter set, since the solver does not have to expend effort looking for solutions beyond the requested gap.
5.17.3.8.2 Solution Pool Subtleties There are a few subtleties associated with finding multiple solutions that we'll cover now.

Continuous Variables

One subtlety arises when considering multiple solutions for models with continuous variables. Specifically, you may have two solutions that take identical values on the integer variables but where some continuous variables differ. By choosing different points on the line between these two solutions, you actually have an infinite number of choices for feasible solutions to the problem. To avoid this issue, we define two solutions as being equivalent if they take the same values on all integer variables (and on all continuous variables that participate in SOS constraints). A solution will be discarded if it is equivalent to another solution that is already in the pool.

Optimality Gap

The interplay between the optimality gap (MIPGap or MIPGapAbs) and multiple solutions can be a bit subtle. When using the default PoolSearchMode, a non-zero optimality gap indicates that you are willing to allow the MIP solver to declare a solution optimal, even though the model may have other, better solutions. The claim the solver makes upon termination is that no other solution would improve the incumbent objective by more than the optimality gap. Terminating at this point is ultimately a pragmatic choice - we'd probably rather have the true best solution, but the cost of reducing the optimality gap to zero can often be prohibitive.

This pragmatic choice can produce a bit of confusion when finding multiple optimal solutions. Specifically, if you ask for the n best solutions, the optimality gap plays a similar role as it does in the default case, but the implications may be a bit harder to understand. Specifically, a non-zero optimality gap means that you are willing to allow the solver to declare that it has found the n best solutions, even though there may be solutions that are better than those that were returned. The claim in this case is that any solution not among the reported n best would improve on the objective for the worst among the n best by less than the optimality gap.

If you want to avoid this source of potential confusion, you should set the optimality gap to 0 when using PoolSearchMode=2.

Logging

If you browse the log from a MIP solve with PoolSearchMode set to a non-default value, you may see the lower bound on the objective exceed the upper bound. This can't happen with the default PoolSearchMode - if you are only looking for one optimal solution, the search is done as soon as the lower bound reaches the upper bound. However, if you are looking for the n best solutions, you have to prove that the model has no solution better than the n-th best. The objective for that n-th solution could be much worse than that of the incumbent. In this situation, the log file will include a line of the form:

Optimal solution found at node 123 - now completing solution pool...

Distributed MIP

One limitation that we should point out related to multiple solutions is that the distributed MIP solver has not been extended to support non-default PoolSearchMode settings. Distributed MIP will typically produce many more feasible solutions than non-distributed MIP, but there's no way to ask it to find the n best solutions.
5.17.3.9 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker’s shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Gurobi provides tools that simplify the task: Gurobi allows you to blend multiple objectives, to treat them hierarchically, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or lexicographic approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective, you only consider solutions that would not degrade the objective values of higher-priority objectives. Gurobi allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Gurobi will only solve multi-objective models with strictly linear objectives. Moreover, for continuous models, Gurobi will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:

```
equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defObj.. obj =e= numShifts - 1/100*sumPreferences;
defNumShifts.. numShifts =e= ...;
defSumPreferences.. sumPreferences =e= ...;

model workforce /all/;
solve workforce minimizing obj using mip;
```

With the default setting GUROBI will solve the blended objective. Using the parameter MultObj GUROBI will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (defObj). In the example, the numShifts objective with coefficient 1 has higher priority than the sumPreferences objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here numShifts will be minimized (same direction as on the solve statement) while sumPreferences will be maximized. GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won’t allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: ObjNRelTol and ObjNAbsTol. By setting one of these for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for numShifts is 100, and if we set ObjNAbsTol for this objective to 20, then the second optimization step maximizing sumPreferences would find the best solution for the second objective from among all solutions with objective 120 or better for numShifts. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).
5.17.4 GAMS Options

The following GAMS options are used by GAMS/Gurobi:

Option BRatio = x;

Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Gurobi not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed. This GAMS option is overridden by the GAMS/Gurobi option UseBasis.

Option IterLim = n;

Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. For MIP problems, if the number of the cumulative simplex iterations exceeds the limit, Gurobi will terminate. This GAMS option is overridden by the GAMS/Gurobi option IterationLimit.

Option NodLim = x;

Maximum number of nodes to process for a MIP problem. This GAMS option is overridden by the GAMS/Gurobi option NodeLimit.

Option OptCA = x;

Absolute optimality criterion for a MIP problem. The OptCA option asks Gurobi to stop when

$$|BP - BF| < \text{OptCA}$$

where $BF$ is the objective function value of the current best integer solution while $BP$ is the best possible integer solution. This GAMS option is overridden by the GAMS/Gurobi option MipGapAbs.

Option OptCR = x;

Relative optimality criterion for a MIP problem. Notice that Gurobi uses a different definition than GAMS normally uses. The OptCR option asks Gurobi to stop when

$$|BP - BF| < |BF| \times \text{OptCR}$$

where $BF$ is the objective function value of the current best integer solution while $BP$ is the best possible integer solution. The GAMS definition is:

$$|BP - BF| < |BP| \times \text{OptCR}$$

This GAMS option is overridden by the GAMS/Gurobi option MipGap.

Option ResLim = x;

Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. Gurobi measures time in wall time on all platforms. Some other GAMS solvers measure time in CPU time on some Unix systems. In case resLim assumes its default value (1e+10) Gurobi will use its own default (infinity). This GAMS option is overridden by the GAMS/Gurobi option TimeLimit.
Option SysOut = On;

Will echo Gurobi messages to the GAMS listing file. This option may be useful in case of a solver failure.

*ModelName*.Cutoff = x;

Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm. This GAMS option is overridden by the GAMS/Gurobi option CutOff.

*ModelName*.OptFile = 1;

Instructs GAMS/Gurobi to read the option file. The name of the option file is *gurobi.opt*.

*ModelName*.PriorOpt = 1;

Instructs GAMS/Gurobi to use the priority branching information passed by GAMS through variable suffix values *variable.prior*.

5.17.5 Summary of GUROBI Options

5.17.5.1 Termination options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bariterlimit</td>
<td>Barrier iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>bestbdstop</td>
<td>Best objective bound to stop</td>
<td>maxdouble</td>
</tr>
<tr>
<td>bestobjstop</td>
<td>Best objective value to stop</td>
<td>mindouble</td>
</tr>
<tr>
<td>cutoff</td>
<td>Objective cutoff</td>
<td>maxdouble</td>
</tr>
<tr>
<td>iterationlim</td>
<td>Simplex iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>memlimit</td>
<td>Memory limit</td>
<td>maxdouble</td>
</tr>
<tr>
<td>nodelimit</td>
<td>MIP node limit</td>
<td>maxdouble</td>
</tr>
<tr>
<td>solutionlim</td>
<td>MIP feasible solution limit</td>
<td>maxint</td>
</tr>
<tr>
<td>timelimit</td>
<td>Time limit</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>worklimit</td>
<td>Work limit</td>
<td>maxdouble</td>
</tr>
</tbody>
</table>

5.17.5.2 Tolerances options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barconvtol</td>
<td>Barrier convergence tolerance</td>
<td>1e-08</td>
</tr>
<tr>
<td>barqcpconvtol</td>
<td>Barrier QCP convergence tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td>feasibilitytol</td>
<td>Primal feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td>intfeastol</td>
<td>Integer feasibility tolerance</td>
<td>1e-05</td>
</tr>
</tbody>
</table>
### 5.17.5.3 Simplex options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpwarmstart</td>
<td>Warm start usage in simplex</td>
<td>1</td>
</tr>
<tr>
<td>normadjust</td>
<td>Simplex pricing norm</td>
<td>-1</td>
</tr>
<tr>
<td>objscale</td>
<td>Objective scaling</td>
<td>0</td>
</tr>
<tr>
<td>perturbvalue</td>
<td>Simplex perturbation magnitude</td>
<td>0.0002</td>
</tr>
<tr>
<td>quad</td>
<td>Quad precision computation in simplex</td>
<td>-1</td>
</tr>
<tr>
<td>scaleflag</td>
<td>Model scaling</td>
<td>-1</td>
</tr>
<tr>
<td>sifting</td>
<td>Sifting within dual simplex</td>
<td>-1</td>
</tr>
<tr>
<td>siftmethod</td>
<td>LP method used to solve sifting sub-problems</td>
<td>-1</td>
</tr>
<tr>
<td>simplexpricing</td>
<td>Simplex variable pricing strategy</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.17.5.4 Barrier options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barcorrectors</td>
<td>Central correction limit</td>
<td>-1</td>
</tr>
<tr>
<td>barhomogeneous</td>
<td>Barrier homogeneous algorithm</td>
<td>-1</td>
</tr>
<tr>
<td>barorder</td>
<td>Barrier ordering algorithm</td>
<td>-1</td>
</tr>
<tr>
<td>crossover</td>
<td>Barrier crossover strategy</td>
<td>-1</td>
</tr>
<tr>
<td>crossoverbasis</td>
<td>Crossover initial basis construction strategy</td>
<td>-1</td>
</tr>
<tr>
<td>qcpdual</td>
<td>Compute dual variables for QCP models</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.17.5.5 MIP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchdir</td>
<td>Branch direction preference</td>
<td>0</td>
</tr>
<tr>
<td>concurrentjobs</td>
<td>Enables distributed concurrent solver</td>
<td>0</td>
</tr>
<tr>
<td>concurrentmip</td>
<td>Enables concurrent MIP solver</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>degenmoves</td>
<td>Degenerate simplex moves</td>
<td>-1</td>
</tr>
<tr>
<td>disconnected</td>
<td>Disconnected component strategy</td>
<td>-1</td>
</tr>
<tr>
<td>distributedmipjobs</td>
<td>Enables the distributed MIP solver</td>
<td>0</td>
</tr>
<tr>
<td>dumpbcsol</td>
<td>Dump incumbents to GDX files during branch-and-cut</td>
<td></td>
</tr>
<tr>
<td>fixoptfile</td>
<td>Option file for fixed problem optimization</td>
<td></td>
</tr>
<tr>
<td>heuristics</td>
<td>Turn MIP heuristics up or down</td>
<td>0.05</td>
</tr>
<tr>
<td>improvstartgap</td>
<td>Trigger solution improvement</td>
<td>0</td>
</tr>
<tr>
<td>improvstartnodes</td>
<td>Trigger solution improvement</td>
<td>maxdouble</td>
</tr>
<tr>
<td>improvstarttime</td>
<td>Trigger solution improvement</td>
<td>maxdouble</td>
</tr>
<tr>
<td>.lazy</td>
<td>Lazy constraints value</td>
<td>0</td>
</tr>
<tr>
<td>lazyconstraints</td>
<td>Indicator to use lazy constraints</td>
<td>0</td>
</tr>
<tr>
<td>minrelnodes</td>
<td>Minimum relaxation heuristic control</td>
<td>-1</td>
</tr>
<tr>
<td>mipfocus</td>
<td>Set the focus of the MIP solver</td>
<td>0</td>
</tr>
<tr>
<td>mipstart</td>
<td>Use mip starting values</td>
<td>0</td>
</tr>
<tr>
<td>mipstopexpr</td>
<td>Stop expression for branch and bound</td>
<td></td>
</tr>
<tr>
<td>miqcpcmetho</td>
<td>Method used to solve MIQCP models</td>
<td>-1</td>
</tr>
<tr>
<td>multiminipstart</td>
<td>Use multiple (partial) mipstarts provided via gdx files</td>
<td></td>
</tr>
<tr>
<td>nlpeur</td>
<td>Controls the NLP heuristic for non-convex quadratic models</td>
<td>1</td>
</tr>
<tr>
<td>nodefiledir</td>
<td>Directory for MIP node files</td>
<td>.</td>
</tr>
<tr>
<td>nodefilestart</td>
<td>Memory threshold for writing MIP tree nodes to disk</td>
<td>maxdouble</td>
</tr>
<tr>
<td>nodemethod</td>
<td>Method used to solve MIP node relaxations</td>
<td>-1</td>
</tr>
<tr>
<td>nonconvex</td>
<td>Control how to deal with non-convex quadratic programs</td>
<td>-1</td>
</tr>
<tr>
<td>norelheurtime</td>
<td>Limits the amount of time (in seconds) spent in the NoRel heuristic</td>
<td>0</td>
</tr>
<tr>
<td>norelheurwork</td>
<td>Limits the amount of work performed by the NoRel heuristic</td>
<td>0</td>
</tr>
<tr>
<td>.partition</td>
<td>Variable partition value</td>
<td>0</td>
</tr>
<tr>
<td>partitionplace</td>
<td>Controls when the partition heuristic runs</td>
<td>15</td>
</tr>
<tr>
<td>.prior</td>
<td>Branching priorities</td>
<td>1</td>
</tr>
<tr>
<td>pumppasses</td>
<td>Feasibility pump heuristic control</td>
<td>-1</td>
</tr>
<tr>
<td>rins</td>
<td>RINS heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>solfiles</td>
<td>Location to store intermediate solution files</td>
<td></td>
</tr>
<tr>
<td>solnpool</td>
<td>Controls export of alternate MIP solutions</td>
<td></td>
</tr>
<tr>
<td>solnpoolmerge</td>
<td>Controls export of alternate MIP solutions for merged GDX solution file</td>
<td></td>
</tr>
<tr>
<td>solnpoolnumsym</td>
<td>Maximum number of variable symbols when writing merged GDX solution file</td>
<td>10</td>
</tr>
</tbody>
</table>
### Solver Manuals

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>solnpoold prefix</td>
<td>First dimension of variables for merged GDX solution file or file name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>solvefixed</td>
<td>Indicator for solving the fixed problem for a MIP to get a dual solution</td>
<td>1</td>
</tr>
<tr>
<td>startnodelimit</td>
<td>Node limit for MIP start sub-MIP</td>
<td>-1</td>
</tr>
<tr>
<td>submipnodes</td>
<td>Nodes explored by sub-MIP heuristics</td>
<td>500</td>
</tr>
<tr>
<td>symmetry</td>
<td>Symmetry detection</td>
<td>-1</td>
</tr>
<tr>
<td>varbranch</td>
<td>Branch variable selection strategy</td>
<td>-1</td>
</tr>
<tr>
<td>zeroobjnodes</td>
<td>Zero objective heuristic control</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.17.5.6 Presolve options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggfill</td>
<td>Allowed fill during presolve aggregation</td>
<td>-1</td>
</tr>
<tr>
<td>aggregate</td>
<td>Presolve aggregation control</td>
<td>1</td>
</tr>
<tr>
<td>dualreductions</td>
<td>Disables dual reductions in presolve</td>
<td>1</td>
</tr>
<tr>
<td>precrush</td>
<td>Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model</td>
<td>0</td>
</tr>
<tr>
<td>predeprow</td>
<td>Presolve dependent row reduction</td>
<td>-1</td>
</tr>
<tr>
<td>predual</td>
<td>Presolve dualization</td>
<td>-1</td>
</tr>
<tr>
<td>premiqcpform</td>
<td>Format of presolved MIQCP model</td>
<td>-1</td>
</tr>
<tr>
<td>prepasses</td>
<td>Presolve pass limit</td>
<td>-1</td>
</tr>
<tr>
<td>preqlinearize</td>
<td>Presolve Q matrix linearization</td>
<td>-1</td>
</tr>
<tr>
<td>Presolve</td>
<td>Presolve level</td>
<td>-1</td>
</tr>
<tr>
<td>presos1bigm</td>
<td>Controls largest coefficient in SOS1 reformulation</td>
<td>-1</td>
</tr>
<tr>
<td>presos1encoding</td>
<td>Controls SOS1 reformulation</td>
<td>-1</td>
</tr>
<tr>
<td>presos2bigm</td>
<td>Controls largest coefficient in SOS2 reformulation</td>
<td>-1</td>
</tr>
<tr>
<td>presos2encoding</td>
<td>Controls SOS2 reformulation</td>
<td>-1</td>
</tr>
<tr>
<td>presparsify</td>
<td>Presolve sparsify reduction</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.17.5.7 Tuning options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tunecleanup</td>
<td>Enables a tuning cleanup phase</td>
<td>0</td>
</tr>
<tr>
<td>tunecriterion</td>
<td>Specify tuning criterion</td>
<td>-1</td>
</tr>
<tr>
<td>tunejobs</td>
<td>Enables distributed tuning</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.17.5.8 MultipleSolutions options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>poolgap</td>
<td>Relative gap for solutions in pool</td>
<td>maxdouble</td>
</tr>
<tr>
<td>poolgapabs</td>
<td>Absolute gap for solutions in pool</td>
<td>maxdouble</td>
</tr>
<tr>
<td>poolsearchmode</td>
<td>Choose the approach used to find additional solutions</td>
<td>0</td>
</tr>
<tr>
<td>poolsize</td>
<td>Number of solutions to keep in pool</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.17.5.9 MIPCuts options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bqcuts</td>
<td>BQP cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cliquecuts</td>
<td>Clique cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>covercuts</td>
<td>Cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cutagggpasses</td>
<td>Constraint aggregation passes performed during cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cutpasses</td>
<td>Root cutting plane pass limit</td>
<td>-1</td>
</tr>
<tr>
<td>cuts</td>
<td>Global cut generation control</td>
<td>-1</td>
</tr>
<tr>
<td>flowcovercuts</td>
<td>Flow cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>flowpathcuts</td>
<td>Flow path cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>gomorypasses</td>
<td>Root Gomory cut pass limit</td>
<td>-1</td>
</tr>
<tr>
<td>gubcovercuts</td>
<td>GUB cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>impliedcuts</td>
<td>Implied bound cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>infproofcuts</td>
<td>Infeasibility proof cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>liftprojectcuts</td>
<td>Lift-and-project cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>mipsepcuts</td>
<td>MIP separation cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>mircuts</td>
<td>MIR cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>modkcuts</td>
<td>Mod-k cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>networkcuts</td>
<td>Network cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>projimpliedcuts</td>
<td>Projected implied bound cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>psdcuts</td>
<td>PSD cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>relaxliftcuts</td>
<td>Relax-and-lift cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>rltcuts</td>
<td>RLT cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>strongcgcuts</td>
<td>Strong-CG cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>submipcuts</td>
<td>Sub-MIP cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>zerohalfcuts</td>
<td>Zero-half cut generation</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.17.5.10 Distributed algorithms options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>workerpassword</td>
<td>Password for distributed worker cluster</td>
<td></td>
</tr>
<tr>
<td>workerpool</td>
<td>Distributed worker cluster</td>
<td></td>
</tr>
</tbody>
</table>

### 5.17.5.11 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayinterval</td>
<td>Frequency at which log lines are printed</td>
<td>5</td>
</tr>
<tr>
<td>.dofuncpieceratio</td>
<td>Control whether to under- or over-estimate function values in PWL approximation</td>
<td>-1</td>
</tr>
<tr>
<td>.dofuncpieces</td>
<td>Sets strategy for PWL function approximation</td>
<td>0</td>
</tr>
<tr>
<td>feasopt</td>
<td>Computes a minimum-cost relaxation to make an infeasible model feasible</td>
<td>0</td>
</tr>
<tr>
<td>feasoptmode</td>
<td>Mode of FeasOpt</td>
<td>0</td>
</tr>
<tr>
<td>feaspref</td>
<td>feasibility preference</td>
<td>1</td>
</tr>
<tr>
<td>feasrelaxbign</td>
<td>Big-M value for feasibility relaxations</td>
<td>1e+06</td>
</tr>
<tr>
<td>fregamsmodel</td>
<td>Preserves memory by dumping the GAMS model instance representation temporarily to disk</td>
<td>0</td>
</tr>
<tr>
<td>funcmaxval</td>
<td>Maximum value for x and y variables in function constraints</td>
<td>1e+06</td>
</tr>
<tr>
<td>funcpiecerror</td>
<td>Error allowed for PWL translation of function constraint</td>
<td>0.001</td>
</tr>
<tr>
<td>funcpieceerror</td>
<td>Piece length for PWL translation of function constraint</td>
<td>0.01</td>
</tr>
<tr>
<td>funcpieceerror</td>
<td>Controls whether to under- or over-estimate function values in PWL approximation</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>funcpieces</td>
<td>Sets strategy for PWL function approximation</td>
<td>0</td>
</tr>
<tr>
<td>.genconstrtype</td>
<td>General constraint type</td>
<td>0</td>
</tr>
<tr>
<td>iis</td>
<td>Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>iismethod</td>
<td>IIS method</td>
<td>-1</td>
</tr>
<tr>
<td>integralityfocus</td>
<td>Set the integrality focus</td>
<td>0</td>
</tr>
<tr>
<td>kappa</td>
<td>Display approximate condition number estimates for the optimal simplex basis</td>
<td>0</td>
</tr>
<tr>
<td>kappaexact</td>
<td>Display exact condition number estimates for the optimal simplex basis</td>
<td>0</td>
</tr>
<tr>
<td>method</td>
<td>Algorithm used to solve continuous models</td>
<td>-1</td>
</tr>
<tr>
<td>miptrace</td>
<td>Filename of MIP trace file</td>
<td></td>
</tr>
<tr>
<td>miptracenode</td>
<td>Node interval when a trace record is written</td>
<td>100</td>
</tr>
<tr>
<td>miptracetime</td>
<td>Time interval when a trace record is written</td>
<td>1</td>
</tr>
<tr>
<td>multiobjmethod</td>
<td>Warm-start method to solve for subsequent objectives</td>
<td>-1</td>
</tr>
<tr>
<td>multiobjpre</td>
<td>Initial presolve on multi-objective models</td>
<td>-1</td>
</tr>
<tr>
<td>multobj</td>
<td>Controls the hierarchical optimization of multiple objectives</td>
<td>0</td>
</tr>
<tr>
<td>names</td>
<td>Indicator for loading names</td>
<td>1</td>
</tr>
<tr>
<td>numericfocus</td>
<td>Set the numerical focus</td>
<td>0</td>
</tr>
<tr>
<td>objnabstol</td>
<td>Allowable absolute degradation for objective</td>
<td></td>
</tr>
<tr>
<td>objnreltol</td>
<td>Allowable relative degradation for objective</td>
<td></td>
</tr>
<tr>
<td>printoptions</td>
<td>List values of all options to GAMS listing file</td>
<td>0</td>
</tr>
<tr>
<td>qextractalg</td>
<td>quadratic extraction algorithm in GAMS interface</td>
<td>0</td>
</tr>
<tr>
<td>readparams</td>
<td>Read Gurobi parameter file</td>
<td></td>
</tr>
<tr>
<td>rerun</td>
<td>Resolve without presolve in case of unbounded or infeasible</td>
<td>0</td>
</tr>
<tr>
<td>rngrestart</td>
<td>Write GAMS readable ranging information file</td>
<td></td>
</tr>
<tr>
<td>seed</td>
<td>Modify the random number seed</td>
<td>0</td>
</tr>
<tr>
<td>sensitivity</td>
<td>Provide sensitivity information</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>Number of parallel threads to use</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>Tuning</td>
<td>Parameter Tuning</td>
<td></td>
</tr>
<tr>
<td>usebasis</td>
<td>Use basis from GAMS</td>
<td>GAMS bratio</td>
</tr>
<tr>
<td>varhint</td>
<td>Guide heuristics and branching through variable hints</td>
<td>0</td>
</tr>
<tr>
<td>writeparams</td>
<td>Write Gurobi parameter file</td>
<td></td>
</tr>
<tr>
<td>writeprob</td>
<td>Save the problem instance</td>
<td></td>
</tr>
</tbody>
</table>
5.17.5.12 The GAMS/Gurobi Options File

The GAMS/Gurobi options file consists of one option or comment per line. An asterisk (\*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file *gurobi.opt*.

```
simplexpricing 3
method 0
```

It will cause Gurobi to use quick-start steepest edge pricing and will use the primal simplex algorithm.

5.17.6 GAMS/Gurobi Log File

Gurobi reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line starts with the iteration number, followed by the objective value, the primal and dual infeasibility values, and the elapsed wall clock time. The dual simplex uses a bigM approach for handling infeasibility, so the objective and primal infeasibility values can both be very large during phase I. The frequency at which log lines are printed is controlled by the DisplayInterval option. By default, the simplex algorithms print a log line roughly every five seconds, although log lines can be delayed when solving models with particularly expensive iterations.

The simplex screen log has the following appearance:

```
Presolve removed 977 rows and 1539 columns
Presolve changed 3 inequalities to equalities
Presolve time: 0.078000 sec.
Presolved: 1748 Rows, 5030 Columns, 32973 Nonzeros

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective</th>
<th>Primal Inf.</th>
<th>Dual Inf.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.8929476e+31</td>
<td>1.200000e+31</td>
<td>1.485042e-04</td>
<td>0s</td>
</tr>
<tr>
<td>5624</td>
<td>1.1486966e+05</td>
<td>0.000000e+00</td>
<td>0.000000e+00</td>
<td>2s</td>
</tr>
</tbody>
</table>

Solved in 5624 iterations and 1.69 seconds
Optimal objective 1.148696610e+05
```

The barrier algorithm log file starts with barrier statistics about dense columns, free variables, nonzeros in \( AA' \) and the Cholesky factor matrix, computational operations needed for the factorization, memory estimate and time estimate per iteration. Then it outputs the progress of the barrier algorithm in iterations with the primal and dual objective values, the magnitude of the primal and dual infeasibilities and the magnitude of the complementarity violation. After the barrier algorithm terminates, by default, Gurobi will perform crossover to obtain a valid basic solution. It first prints the information about pushing the dual and primal superbasic variables to the bounds and then the information about the simplex progress until the completion of the optimization.

The barrier screen log has the following appearance:
Presolve removed 2394 rows and 3412 columns
Presolve time: 0.09s
Presolved: 3677 Rows, 8818 Columns, 30934 Nonzeros
Ordering time: 0.20s

Barrier statistics:
Dense cols : 10
Free vars : 3
AA’ NZ : 9.353e+04
Factor NZ : 1.139e+06 (roughly 14 MBytes of memory)
Factor Ops : 7.388e+08 (roughly 2 seconds per iteration)

<table>
<thead>
<tr>
<th>Objective</th>
<th>Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Primal</td>
</tr>
<tr>
<td>0</td>
<td>1.11502515e+13</td>
</tr>
<tr>
<td>1</td>
<td>4.40523949e+12</td>
</tr>
<tr>
<td>2</td>
<td>1.18016996e+12</td>
</tr>
<tr>
<td>3</td>
<td>2.24963388e+11</td>
</tr>
<tr>
<td>4</td>
<td>4.63336675e+10</td>
</tr>
<tr>
<td>5</td>
<td>1.25266057e+10</td>
</tr>
<tr>
<td>6</td>
<td>1.53128732e+09</td>
</tr>
<tr>
<td>7</td>
<td>5.70973983e+08</td>
</tr>
<tr>
<td>8</td>
<td>2.91558669e+08</td>
</tr>
<tr>
<td>9</td>
<td>1.22383252e+08</td>
</tr>
<tr>
<td>10</td>
<td>6.47115867e+07</td>
</tr>
</tbody>
</table>

Barrier solved model in 27 iterations and 1.86 seconds
Optimal objective 1.12663961e+07

Crossover log...

1592 DPUSHES remaining with DInf 0.0000000e+00 2s
0 DPUSHES remaining with DInf 2.8167333e-06 2s

180 PPUSHES remaining with PInf 0.0000000e+00 2s
0 PPUSHES remaining with PInf 0.0000000e+00 2s

Push phase complete: Pinf 0.0000000e+00, Dinf 2.8167333e-06 2s

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective</th>
<th>Primal Inf.</th>
<th>Dual Inf.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1776</td>
<td>1.1266396e+07</td>
<td>0.0000000e+00</td>
<td>0.0000000e+00</td>
<td>2s</td>
</tr>
</tbody>
</table>

Solved in 2043 iterations and 2.00 seconds
Optimal objective 1.126639605e+07

For MIP problems, the Gurobi solver prints regular status information during the branch and bound search. The first two output columns in each log line show the number of nodes that have been explored so far in the search tree, followed by the number of nodes that remain unexplored. The next three columns provide information on the most recently explored node in the tree. The solver prints the relaxation objective value for this node, followed by its depth in the search tree, followed by the number of integer variables with fractional values in the node relaxation solution. The next three columns provide information on the progress of the global MIP bounds. They show the objective value for the best known integer feasible solution, the best bound on the value of the optimal solution, and the gap between these lower and upper bounds. Finally, the last two columns provide information on the amount of work performed so far. The
first column gives the average number of simplex iterations per explored node, and the next column gives
the elapsed wall clock time since the optimization began.

At the default value for option DisplayInterval, the MIP solver prints one log line roughly every five
seconds. Note, however, that log lines are often delayed in the MIP solver due to particularly expensive
nodes or heuristics.

Presolve removed 12 rows and 11 columns
Presolve tightened 70 bounds and modified 235 coefficients
Presolve time: 0.02s
Presolved: 114 Rows, 116 Columns, 424 Nonzeros
Objective GCD is 1

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Expl Unexpl</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>H 0</td>
<td>0</td>
<td>-0.0000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Root relaxation: 208 iterations, 0.00 seconds</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>29.6862</td>
<td>0 64</td>
<td>-0.0000 29.6862</td>
<td>-</td>
</tr>
<tr>
<td>H 0</td>
<td>0</td>
<td>8.0000</td>
<td>29.6862 271%</td>
<td>-</td>
</tr>
<tr>
<td>H 0</td>
<td>0</td>
<td>17.0000</td>
<td>29.6862 74.6%</td>
<td>-</td>
</tr>
<tr>
<td>0 2</td>
<td>27.4079</td>
<td>0 60</td>
<td>17.0000 27.4079</td>
<td>61.2%</td>
</tr>
<tr>
<td>H 27</td>
<td>17</td>
<td>18.0000</td>
<td>26.0300 44.6%</td>
<td>51.6</td>
</tr>
<tr>
<td>* 87</td>
<td>26</td>
<td>45</td>
<td>20.0000 26.0300</td>
<td>30.2%</td>
</tr>
<tr>
<td>* 353</td>
<td>71</td>
<td>29</td>
<td>21.0000 25.0000</td>
<td>19.0%</td>
</tr>
<tr>
<td>1268</td>
<td>225</td>
<td>24.0000</td>
<td>28 43</td>
<td>21.0000 24.0000 14.3% 32.3</td>
</tr>
<tr>
<td>2215</td>
<td>464</td>
<td>22.0000</td>
<td>43 30</td>
<td>21.0000 24.0000 14.3% 33.2</td>
</tr>
</tbody>
</table>

Cutting planes:
Gomory: 175
Cover: 25
Implied bound: 87
MIR: 150

Explored 2550 nodes (84600 simplex iterations) in 11.67 seconds
Thread count was 1 (of 4 available processors)

Optimal solution found (tolerance 1.00e-01)
Best objective 2.1000000000e+01, best bound 2.3000000000e+01, gap 9.5238%

5.17.7 Detailed Descriptions of GUROBI Options

aggfill (integer): Allowed fill during presolve aggregation

Controls the amount of fill allowed during presolve aggregation. Larger values generally lead
to presolved models with fewer rows and columns, but with more constraint matrix non-zeros.

The default value chooses automatically, and usually works well.

Default: -1

aggregate (integer): Presolve aggregation control

Default: 1

barconvtol (real): Barrier convergence tolerance


The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance.

Default: $1e^{-08}$

**barcorrectors** (integer): Central correction limit

The default value is chosen automatically, depending on problem characteristics.

Default: $-1$

**barhomogeneous** (integer): Barrier homogeneous algorithm

Determines whether to use the homogeneous barrier algorithm. At the default setting (-1), it is only used when barrier solves a node relaxation for a MIP model. Setting the parameter to 0 turns it off, and setting it to 1 forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness. It is a bit slower than the default algorithm.

Default: $-1$

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Homogeneous Barrier off</td>
</tr>
<tr>
<td>1</td>
<td>Force Homogeneous Barrier on</td>
</tr>
</tbody>
</table>

**bariterlimit** (integer): Barrier iteration limit

Default: infinity

**barorder** (integer): Barrier ordering algorithm

Default: $-1$

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Approximate Minimum Degree ordering</td>
</tr>
<tr>
<td>1</td>
<td>Nested Dissection ordering</td>
</tr>
</tbody>
</table>

**barqcpcrontol** (real): Barrier QCP convergence tolerance

When solving a QCP model, the barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Tightening this tolerance may lead to a more accurate solution, but it may also lead to a failure to converge.

Default: $1e^{-06}$

**bestbdstop** (real): Best objective bound to stop

Terminates as soon as the engine determines that the best bound on the objective value is at least as good as the specified value.

Default: maxdouble

**bestobjstop** (real): Best objective value to stop
Terminate as soon as the engine finds a feasible solution whose objective value is at least as good as the specified value.

Default: mindouble

**bqpcuts (integer):** BQP cut generation

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**branchdir (integer):** Branch direction preference

This option allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, this parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Always explore the down branch first</td>
</tr>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>Always explore the up branch first</td>
</tr>
</tbody>
</table>

**cliquecuts (integer):** Clique cut generation

See the description of the global Cuts parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**concurrentjobs (integer):** Enables distributed concurrent solver

Enables distributed concurrent optimization, which can be used to solve LP or MIP models on multiple machines. A value of $n$ causes the solver to create $n$ independent models, using different parameter settings for each. Each of these models is sent to a distributed worker for processing. Optimization terminates when the first solve completes. Use the WorkerPool parameter to provide a list of available distributed workers.

By default, Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search.
By bringing the resources of multiple machines to bear on a single model, this approach can sometimes solve models much faster than a single machine.

Default: 0

**concurrentmip** *(integer)*: Enables concurrent MIP solver

This parameter enables the concurrent MIP solver. When the parameter is set to value $n$, the MIP solver performs $n$ independent MIP solves in parallel, with different parameter settings for each. Optimization terminates when the first solve completes. Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. This approach can sometimes solve models much faster than applying all available threads to a single MIP solve, especially on very large parallel machines.

The concurrent MIP solver divides available threads evenly among the independent solves. For example, if you have 6 threads available and you set ConcurrentMIP to 2, the concurrent MIP solver will allocate 3 threads to each independent solve. Note that the number of independent solves launched will not exceed the number of available threads.

The concurrent MIP solver produces a slightly different log from the standard MIP solver. The log only provides periodic summary information. Each concurrent MIP log line shows the objective for the best feasible solution found by any of the independent solves to that point, the best objective bound proved by any of the independent solves, and the relative gap between these two values. Gurobi also includes node counts from one of the independent solves, as well as elapsed times, to give some indication of forward progress.

Default: 1

**covercuts** *(integer)*: Cover cut generation

See the description of the global Cuts parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**crossover** *(integer)*: Barrier crossover strategy

Use value 0 to disable crossover; the solver will return an interior solution. Other options control whether the crossover algorithm tries to push primal or dual variables to bounds first, and then which simplex algorithm is used once variable pushing is complete. Options 1 and 2 push dual variables first, then primal variables. Option 1 finishes with primal, while option 2 finishes with dual. Options 3 and 4 push primal variables first, then dual variables. Option 3 finishes with primal, while option 4 finishes with dual. The default value of -1 chooses automatically.

Default: -1

**crossoverbasis** *(integer)*: Crossover initial basis construction strategy

Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Chooses an initial basis quickly</td>
</tr>
<tr>
<td>1</td>
<td>Can take much longer, but often produces a more numerically stable start basis</td>
</tr>
</tbody>
</table>

**cutaggpasses** *(integer)*: Constraint aggregation passes performed during cut generation

A non-negative value indicates the maximum number of constraint aggregation passes performed during cut generation. See the description of the global Cuts parameter for further information.

Default: -1

**cutoff** *(real)*: Objective cutoff

Optimization will terminate if the engine determines that the optimal objective value for the model is worse than the specified cutoff. This option overwrites the GAMS cutoff option.

Default: maxdouble

**cutpasses** *(integer)*: Root cutting plane pass limit

Default: -1

**cuts** *(integer)*: Global cut generation control

The parameters, Cuts, CliqueCuts, CoverCuts, FlowCoverCuts, FlowPathCuts, GUBCoverCuts, ImpliedCuts, InfProofCuts, MIPSepCuts, MIRCuts, ModKCuts, NetworkCuts, GomoryPasses, StrongCGCuts, SubMIPCuts, CutAggPasses and ZeroHalfCuts, affect the generation of MIP cutting planes. In all cases except GomoryPasses and CutAggPasses, a value of -1 corresponds to an automatic setting, which allows the solver to determine the appropriate level of aggressiveness in the cut generation. Unless otherwise noted, settings of 0, 1, and 2 correspond to no cut generation, conservative cut generation, or aggressive cut generation, respectively. The Cuts parameter provides global cut control, affecting the generation of all cuts. This parameter also has a setting of 3, which corresponds to very aggressive cut generation. The other parameters override the global cuts parameter (so setting Cuts to 2 and CliqueCuts to 0 would generate all cut types aggressively, except clique cuts which would not be generated at all. Setting Cuts to 0 and GomoryPasses to 10 would not generate any cuts except Gomory cuts for 10 passes).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
<tr>
<td>3</td>
<td>Very aggressive</td>
</tr>
</tbody>
</table>

**degenmoves** *(integer)*: Degenerate simplex moves

Limits degenerate simplex moves. These moves are performed to improve the integrality of the current relaxation solution. By default, the algorithm chooses the number of moves to perform automatically.
Changing the value of this parameter can help performance in cases where an excessive amount of time is spent after the initial root relaxation has been solved but before the cut generation process or the root heuristics have started.

Default: -1

**disconnected (integer)**: Disconnected component strategy

A MIP model can sometimes be made up of multiple, completely independent sub-models. This parameter controls how aggressively we try to exploit this structure. A value of 0 ignores this structure entirely, while larger values try more aggressive approaches. The default value of -1 chooses automatically. This only affects mixed integer programming (MIP) models.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Ignores structure entirely</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**displayinterval (integer)**: Frequency at which log lines are printed

Default: 5

**distributedmipjobs (integer)**: Enables the distributed MIP solver

Enables distributed MIP. A value of \( n \) causes the MIP solver to divide the work of solving a MIP model among \( n \) machines. Use the `WorkerPool` parameter to provide the list of available machines.

Default: 0

**.dofuncpieceerror (real)**: Error allowed for PWL translation of function constraints

Default: 1e-3

**.dofuncpiecelength (real)**: Piece length for PWL translation of function constraints

The default length behavior for piecewise-linear approximation of a function constraint is controlled by `funcPieceLength`. This dot option `.dofuncPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: 1e-2

**.dofuncpieceratio (real)**: Control whether to under- or over-estimate function values in PWL approximation

The default ratio behavior for piecewise-linear approximation of a function constraint is controlled by `funcPieceRatio`. This dot option `.dofuncPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: -1

**.dofuncpieces (integer)**: Sets strategy for PWL function approximation

The dot option `.dofuncpieces` allows to specify the strategy for piecewise-linear approximation of function constraints. This option is particularly useful when working with piecewise-linear functions in MIP models.

Default: -1
The default strategy for performing a piecewise-linear approximation of a function constraint is set by `funcPieces`. This dot option .do`funcPieces` allows to overwrite the default strategy by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

`dofuncpieceerror` The default error behavior for piecewise-linear approximation of a function constraint is controlled by `funcPieceError`. This dot option .do`funcPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Bounds the relative error of the approximation; the error bound is provided in the <code>FuncPieceError</code> parameter</td>
</tr>
<tr>
<td>-1</td>
<td>Bounds the absolute error of the approximation; the error bound is provided in the <code>FuncPieceError</code> parameter</td>
</tr>
<tr>
<td>0</td>
<td>Use value of <code>FuncPieces</code></td>
</tr>
<tr>
<td>1</td>
<td>Uses a fixed width for each piece; the actual width is provided in the <code>FuncPieceLength</code> parameter</td>
</tr>
<tr>
<td>&gt;=2</td>
<td>Sets the number of pieces; pieces are equal width</td>
</tr>
</tbody>
</table>

`dualreductions` (boolean): Disables dual reductions in presolve

Default: 1

`dumpbcsol` (string): Dump incumbents to GDX files during branch-and-cut

The content of this string option is used as a file stem for GDX point files. The file name gets completed by the solution number. So if this option has been set to `mysol` then GDX files containing the new solution with the names `mysol0.gdx`, `mysol1.gdx`, ... will be created. The number of GDX files created depends on the number of solutions Gurobi finds during branch-and-cut.

`feasibilitytol` (real): Primal feasibility tolerance

All constraints must be satisfied to a tolerance of `FeasibilityTol`.

Range: \([1e-09, 0.01]\]

Default: \(1e-06\)

`feasopt` (boolean): Computes a minimum-cost relaxation to make an infeasible model feasible

With `Feasopt` turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option `FeasOptMode`.

Feasible relaxations are available for all problem types.

Default: 0
feasoptmode (integer): Mode of FeasOpt

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimize sum of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of all required relaxations in first phase only</td>
</tr>
<tr>
<td>1</td>
<td>Minimize sum of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
<tr>
<td>2</td>
<td>Minimize number of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the number of constraints and bounds requiring relaxation in first phase only</td>
</tr>
<tr>
<td>3</td>
<td>Minimize number of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
<tr>
<td>4</td>
<td>Minimize sum of squares of relaxations</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of squares of required relaxations in first phase only</td>
</tr>
<tr>
<td>5</td>
<td>Minimize sum of squares of relaxations and optimize</td>
</tr>
<tr>
<td></td>
<td>Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations</td>
</tr>
</tbody>
</table>

.feaspref (real): feasibility preference

You can express the costs associated with relaxing a bound or right hand side value during a FeasOpt run through the .feaspref option. The syntax for dot options is explained in the Introduction chapter of the Solver Manual. The input value denotes the user’s willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: 1

feasrelaxbigm (real): Big-M value for feasibility relaxations

When relaxing a constraint in a feasibility relaxation, it is sometimes necessary to introduce a big-M value. This parameter determines the default magnitude of that value.

Default: 1e+06
**fixoptfile** (string): Option file for fixed problem optimization

**flowcovercuts** (integer): Flow cover cut generation

See the description of the global Cuts parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**flowpathcuts** (integer): Flow path cut generation

See the description of the global Cuts parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**freegamsmodel** (boolean): Preserves memory by dumping the GAMS model instance representation temporarily to disk

In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with SolveLink=0 and only for models without quadratic constraints.

Default: 0

**funcmaxval** (real): Maximum value for x and y variables in function constraints

Very large values in piecewise-linear approximations can cause numerical issues. This parameter limits the bounds on the variables that participate in function constraints. Specifically, if x or y participate in a function constraint, any bound larger than funcMaxVal (in absolute value) will be truncated.

Default: 1e+06

**funcpieceerror** (real): Error allowed for PWL translation of function constraint

If the **funcPieces** parameter is set to value -1 or -2, this parameter provides the maximum allowed error (absolute for -1, relative for -2) in the piecewise-linear approximation.

Default: 0.001

**funcpiecelength** (real): Piece length for PWL translation of function constraint
If the `funcPieces` parameter is set to value 1, this parameter gives the length of each piece of the piecewise-linear approximation.

Default: 0.01

`funcpieceratio (real)`: Controls whether to under- or over-estimate function values in PWL approximation

This option controls whether the piecewise-linear approximation of a function constraint is an underestimate of the function, an overestimate, or somewhere in between. A value of 0 will always underestimate, while a value of 1 will always overestimate. A value in between will interpolate between the underestimate and the overestimate. A special value of -1 chooses points that are on the original function.

Default: -1

`funcpieces (integer)`: Sets strategy for PWL function approximation

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Bounds the relative error of the approximation; the error bound is provided in the FuncPieceError parameter</td>
</tr>
<tr>
<td>-1</td>
<td>Bounds the absolute error of the approximation; the error bound is provided in the FuncPieceError parameter</td>
</tr>
<tr>
<td>0</td>
<td>Automatic PWL approximation</td>
</tr>
<tr>
<td>1</td>
<td>Uses a fixed width for each piece; the actual width is provided in the FuncPieceLength parameter</td>
</tr>
<tr>
<td>&gt;1</td>
<td>Sets the number of pieces; pieces are equal width</td>
</tr>
</tbody>
</table>

`.genconstrtype (integer)`: General constraint type

Mathematical programming has traditionally defined a set of fundamental constraint types: variable bound constraints, linear constraints, quadratic constraints, integrality constraints, and SOS constraints. These are typically treated directly by the underlying solver (although not always), and are fundamental to the overall algorithm.

Gurobi accepts a number of additional constraint types, which we collectively refer to as general constraints. These are typically not treated directly by the solver. Rather, they are transformed by presolve into mathematically equivalent sets of constraints (and variables), chosen from among the fundamental types listed above. These general constraints are provided as a convenience to users. If such constraints appear in your model, but if you prefer to reformulate them yourself using fundamental constraint types instead, you can certainly do so. However, note that Gurobi can sometimes exploit information contained in the other constraints in the model to build a more efficient formulation than what you might create. Gurobi does not allow general logical constraints but needs the constraint with a single operator (either `max` or `min` or ...). In order to pass the general constraint type to Gurobi we interpret a linear constraint plus the value of the `GenConstrType` option. Such a linear constraint has a resultant variable (r or rb) on the left hand side of `=`E=`) and the variables involved in the operation on the right hand side:

**MAX constraint:**

`eqMax.. r =e= max(x1,x2,x3,c);`

is represented in the following form:
eqMax.. r =e= x1+x2+x3+c;
with GAMS/GUROBI option eqMax.GenConstrType=1.

MIN constraint:
eqMin.. r =e= min(x1,x2,x3,c);
is represented in the following form:
eqMin.. r =e= x1+x2+x3+c;
with GAMS/GUROBI option eqMin.GenConstrType=2.

ABS constraint:
eqAbs.. r =e= abs(x1);
is represented in the following form:
eqAbs.. r =e= x1;
with GAMS/GUROBI option eqAbs.GenConstrType=3.

AND constraint:
eqAnd.. br =e= b1 and b2 and b3 and b4;
is represented in the following form:
eqAnd.. br =e= b1+b2+b3+b4;
with GAMS/GUROBI option eqAnd.GenConstrType=4.

OR constraint:
eqOr.. br =e= b1 or b2 or b3 or b4;
is represented in the following form:
eqOr.. br =e= b1+b2+b3+b4;
with GAMS/GUROBI option eqOr.GenConstrType=5.

EXPA constraint:
eqExpA.. r =e= a**x1;
is represented in the following form:
eqExpA.. r =e= x1+a;
with GAMS/GUROBI option eqExpA.GenConstrType=6.

LOGA constraint:
eqLogA.. r =e= log(x1)/log(a);
is represented in the following form:
eqLogA.. r =e= x1+a;
with GAMS/GUROBI option eqLogA.GenConstrType=7.

POW constraint:

eqPow.. r =e= x1**a;

is represented in the following form:

eqPow.. r =e= x1+a;

with GAMS/GUROBI option eqPow.GenConstrType=8.

EXP constraint:

eqExp.. r =e= exp(x1);

is represented in the following form:

eqExp.. r =e= x1;

with GAMS/GUROBI option eqExp.GenConstrType=9.

LOG constraint:

eqLog.. r =e= log(x1);

is represented in the following form:

eqLog.. r =e= x1;

with GAMS/GUROBI option eqLog.GenConstrType=10.

SIN constraint:

eqSin.. r =e= sin(x1);

is represented in the following form:

eqSin.. r =e= x1;

with GAMS/GUROBI option eqSin.GenConstrType=11.

COS constraint:

eqCos.. r =e= cos(x1);

is represented in the following form:

eqCos.. r =e= x1;

with GAMS/GUROBI option eqCos.GenConstrType=12.

TAN constraint:

eqTan.. r =e= tan(x1);

is represented in the following form:
eqTan.. r =e= x1;

0-NORM constraint:

eqNorm0.. r =e= norm0(x1,x2,x3);
is represented in the following form:

eqNorm0.. r =e= x1 + x2 + x3;
with GAMS/GUROBI option eqNorm0.GenConstrType=14.

1-NORM constraint:

eqNorm1.. r =e= norm1(x1,x2,x3);
is represented in the following form:

eqNorm1.. r =e= x1 + x2 + x3;
with GAMS/GUROBI option eqNorm1.GenConstrType=15.

2-NORM constraint:

eqNorm2.. r =e= norm2(x1,x2,x3);
is represented in the following form:

eqNorm2.. r =e= x1 + x2 + x3;
with GAMS/GUROBI option eqNorm2.GenConstrType=16. Note that a 2-norm constraint can lead to a non-convex quadratic model which is much harder to solve than a convex quadratic or linear model.

INF-NORM constraint:

eqNormInf.. r =e= normInf(x1,x2,x3);
is represented in the following form:

eqNormInf.. r =e= x1 + x2 + x3;
with GAMS/GUROBI option eqNormInf.GenConstrType=17.

Note that for the AND and OR constraint the resultant variable needs to be a binary variable.
The default value for parameter GenConstrType of a constraint is 0 marking it as a regular constraint (linear or quadratic).
Default: 0

gomorypasses (integer): Root Gomory cut pass limit ←

A non-negative value indicates the maximum number of Gomory cut passes performed. See the description of the global Cuts parameter for further information.

Default: -1

gubcovercuts (integer): GUB cover cut generation ←

See the description of the global Cuts parameter for further information.

Default: -1
heuristics (real): Turn MIP heuristics up or down

Larger values produce more and better feasible solutions, at a cost of slower progress in the best bound.

Range: [0, 1]

Default: 0.05

iis (boolean): Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible

Default: 0

iismethod (integer): IIS method

Chooses the IIS method to use. Method 0 is often faster, while method 1 can produce a smaller IIS. Method 2 ignores the bound constraints. Method 3 will return the IIS for the LP relaxation of a MIP model if the relaxation is infeasible, even though the result may not be minimal when integrality constraints are included. The default value of -1 chooses automatically.

Default: -1

impliedcuts (integer): Implied bound cut generation

See the description of the global Cuts parameter for further information.

Default: -1

improvestartgap (real): Trigger solution improvement

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify an optimality gap at which the MIP solver will switch to this strategy. For example, setting this parameter to 0.1 will cause the MIP solver to switch once the relative optimality gap is smaller than 0.1.

Default: 0
\textbf{improvestartnodes (real):} Trigger solution improvement $\leftrightarrow$

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify the node count at which the MIP solver switches to a solution improvement strategy. For example, setting this parameter to 10 will cause the MIP solver to switch strategies once the node count is larger than 10.

Default: \texttt{maxdouble}

\textbf{improvestarttime (real):} Trigger solution improvement $\leftrightarrow$

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify a time limit when the MIP solver will switch to this strategy. For example, setting this parameter to 10 will cause the MIP solver to switch 10 seconds after starting the optimization.

Default: \texttt{maxdouble}

\textbf{infproofcuts (integer):} Infeasibility proof cut generation $\leftrightarrow$

Controls infeasibility proof cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the \texttt{Cuts} parameter.

Default: -1

\textbf{integralityfocus (boolean):} Set the integrality focus $\leftrightarrow$

One unfortunate reality in MIP is that integer variables don’t always take exact integral values. While this typically doesn’t create significant problems, in some situations the side-effects can be quite undesirable. The best-known example is probably a trickle flow, where a continuous variable that is meant to be zero when an associated binary variable is zero instead takes a non-trivial value. More precisely, given a constraint \( y = 1 \leq \M b \), where \( y \) is a non-negative continuous variable, \( b \) is a binary variable, and \( M \) is a constant that captures the largest possible value of \( y \), the constraint is intended to enforce the relationship that \( y \) must be zero if \( b \) is zero. With the default integer feasibility tolerance, the binary variable is allowed to take a value as large as \( 1e^{-5} \) while still being considered as taking value zero. If the \( M \) value is large, then the \( M b \) upper bound on the \( y \) variable can be substantial.

Reducing the value of the \texttt{intFeasTol} parameter can mitigate the effects of such trickle flows, but often at a significant cost, and often with limited success. The \texttt{integralityFocus} parameter provides a better alternative. Setting this parameter to 1 requests that the solver work harder to try to avoid solutions that exploit integrality tolerances. More precisely, the solver tries to find solutions that are still (nearly) feasible if all integer variables are rounded to exact integral values. We should say that the solver won’t always succeed in finding such solutions, and that this setting introduces a modest performance penalty, but the setting will significantly reduce the frequency and magnitude of such violations.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable the integrality focus</td>
</tr>
<tr>
<td>1</td>
<td>Enable the integrality focus</td>
</tr>
</tbody>
</table>
intfeastol (real): Integer feasibility tolerance

An integrality restriction on a variable is considered satisfied when the variable's value is less than IntFeasTol from the nearest integer value.

Range: [1e-09, 0.1]
Default: 1e-05

iterationlimit (real): Simplex iteration limit

Default: infinity

kappa (boolean): Display approximate condition number estimates for the optimal simplex basis

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not compute and display approximate condition number</td>
</tr>
<tr>
<td>1</td>
<td>Compute and display approximate condition number</td>
</tr>
</tbody>
</table>

kappaexact (boolean): Display exact condition number estimates for the optimal simplex basis

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not compute and display exact condition number</td>
</tr>
<tr>
<td>1</td>
<td>Compute and display exact condition number</td>
</tr>
</tbody>
</table>

.lazy (integer): Lazy constraints value

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1, 2, or 3 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraint, the solution is discarded and one or more of the violated lazy constraints are pulled into the active model.

Larger values for this attribute cause the constraint to be pulled into the model more aggressively. With a value of 1, the constraint can be used to cut off a feasible solution, but it won't necessarily be pulled in if another lazy constraint also cuts off the solution. With a value of 2, all lazy constraints that are violated by a feasible solution will be pulled into the model. With a value of 3, lazy constraints that cut off the relaxation solution are also pulled in.

Any constraint whose Lazy attribute is set to -1 is treated as a user cut; it is removed from the model and placed in the user cut pool. User cuts may be added to the model at any node in the branch-and-cut search tree to cut off relaxation solutions.

The main difference between user cuts and lazy constraints is that the former are not allowed to cut off integer-feasible solutions. In other words, they are redundant for the MIP model, and the solver is free to decide whether or not to use them to cut off relaxation solutions. The hope is that adding them speeds up the overall solution process. Lazy constraints have no such restrictions. They are essential to the model, and the solver is forced to apply them whenever a solution would otherwise not satisfy them.
Only affects MIP models. Lazy constraints are only active if option LazyConstraints is enabled and are specified through the option lazy. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: 0

lazyconstraints (boolean): Indicator to use lazy constraints

Default: 0

liftprojectcuts (integer): Lift-and-project cut generation

Controls lift-and-project cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: -1

lpwarmstart (integer): Warm start usage in simplex

Controls whether and how Gurobi uses warm start information for an LP optimization. The non default setting of 2 is particularly useful for communicating advanced start information while retaining the performance benefits of presolve.

As a general rule, setting this parameter to 0 ignores any start information and solves the model from scratch. Setting it to 1 (the default) uses the provided warm start information to solve the original, unpresolved problem, regardless of whether presolve is enabled. Setting it to 2 uses the start information to solve the presolved problem, assuming that presolve is enabled. This involves mapping the solution of the original problem into an equivalent (or sometimes nearly equivalent) crushed solution of the presolved problem. If presolve is disabled, then setting 2 still prioritizes start vectors, while setting 1 prioritizes basis statuses. Taken together, the LPWarmStart parameter setting, the LP algorithm specified by Gurobi's Method parameter, and the available advanced start information determine whether Gurobi will use basis statuses only, basis statuses augmented with information from start vectors, or a basis obtained by applying the crossover method to the provided primal and dual start vectors to jump start the optimization.

When Gurobi's Method parameter requests the barrier solver, primal and dual start vectors are prioritized over basis statuses (but only if you provide both). These start vectors are fed to the crossover procedure. This is the same crossover that is used to compute a basic solution from the interior solution produced by the core barrier algorithm, but in this case crossover is started from arbitrary start vectors. If you set the LPWarmStart parameter to 1, crossover will be invoked on the original model using the provided vectors. Any provided basis information will not be used in this case. If you set LPWarmStart to 2, crossover will be invoked on the presolved model using crushed start vectors. If you set the parameter to 2 and provide a basis but no start vectors, the basis will be used to compute the corresponding primal and dual solutions on the original model. Those solutions will then be crushed and used as primal and dual start vectors for the crossover, which will then construct a basis for the presolved model. Note that for all of these settings and start combinations, no barrier algorithm iterations are performed.

The simplex algorithms provide more warm-starting options, with a parameter value of 1, simplex will start from a provided basis, if available. Otherwise, it uses a provided start vector to refine the crash basis it computes.

With a value of 2, simplex will use the crushed start vector on the presolved model to refine the crash basis. This is true regardless of whether the start is derived from start vectors or a starting basis from the original model. The difference is that if you provide an advanced basis, the basis will be used to compute the corresponding primal and dual solutions on the original model from which the primal or dual start on the presolved model will be derived.

Note: Only affects linear programming (LP) models.

Default: 1
**markowitztol** *(real)*: Threshold pivoting tolerance

Used to limit numerical error in the simplex algorithm. A larger value may avoid numerical problems in rare situations, but it will also harm performance.

Range: [0.0001, 0.999]

Default: 0.0078125

**memlimit** *(real)*: Memory limit

Limits the total amount of memory (in GB, i.e., $10^9$ bytes) available to Gurobi. If more is needed, Gurobi will fail with an OUT_OF_MEMORY error.

Default: maxdouble

**method** *(integer)*: Algorithm used to solve continuous models

Synonyms: lpmethod rootmethod

Algorithm used to solve continuous models or the root node of a MIP model. Options are:
-1=automatic, 0=primal simplex, 1=dual simplex, 2=barrier, 3=concurrent, 4=deterministic concurrent, 5=deterministic concurrent simplex.

In the current release, the default Automatic (-1) setting will typically choose non-deterministic concurrent (Method=3) for an LP, barrier (Method=2) for a QP or QCP, and dual (Method=1) for the MIP root node. Only the simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Method=3 and Method=4 will run dual simplex, barrier, and sometimes primal simplex (depending on the number of available threads). Method=5 will run both primal and dual simplex. The deterministic options (Method=4 and Method=5) give the exact same result each time, while Method=3 is often faster but can produce different optimal bases when run multiple times.

The default setting is rarely significantly slower than the best possible setting, so you generally won’t see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider using the dual simplex method (Method=1). The concurrent optimizer, which is typically chosen when using the default setting, consumes a lot more memory than dual simplex alone.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>1</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>2</td>
<td>Barrier</td>
</tr>
<tr>
<td>3</td>
<td>Concurrent</td>
</tr>
<tr>
<td>4</td>
<td>Deterministic concurrent</td>
</tr>
<tr>
<td>5</td>
<td>Both primal and dual simplex</td>
</tr>
</tbody>
</table>
minrelnodes (integer): Minimum relaxation heuristic control

Number of nodes to explore in the minimum relaxation heuristic. Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

The default value automatically chooses whether to apply the heuristic. It will only rarely choose to do so.

Default: $-1$

mipfocus (integer): Set the focus of the MIP solver

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Balance between finding good feasible solutions and proving optimality</td>
</tr>
<tr>
<td>1</td>
<td>Focus towards finding feasible solutions</td>
</tr>
<tr>
<td>2</td>
<td>Focus towards proving optimality</td>
</tr>
<tr>
<td>3</td>
<td>Focus on moving the best objective bound</td>
</tr>
</tbody>
</table>

mipgap (real): Relative MIP optimality gap

The MIP engine will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than $\text{MipGap}$ times the upper bound.

Range: $[0, \infty]$  

Default: GAMS optcr

mipgapabs (real): Absolute MIP optimality gap

The MIP solver will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than $\text{MIPGapAbs}$.

Range: $[0, \infty]$  

Default: GAMS optca

mipsepcuts (integer): MIP separation cut generation

See the description of the global Cuts parameter for further information.

Default: $-1$

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1$</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>
**mipstart (boolean):** Use mip starting values

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use the values</td>
</tr>
<tr>
<td>1</td>
<td>Use the values</td>
</tr>
</tbody>
</table>

**mipstopexpr (string):** Stop expression for branch and bound

If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: resusd, nodusd, objest, objval. Supported operators are: +, -, *, /, ^, %, !=, ==, <=, >=, >, ==, !, &&, ||, (, ), abs, ceil, exp, floor, log, log10, pow, sqrt. Example: nodusd >= 1000 && abs(objest - objval) / abs(objval) < 0.1

**miptrace (string):** Filename of MIP trace file

More info is available in chapter Solve trace.

**miptracenode (integer):** Node interval when a trace record is written

Default: 100

**miptracetime (real):** Time interval when a trace record is written

Default: 1

**miqcpmethod (integer):** Method used to solve MIQCP models

Controls the method used to solve MIQCP models. Value 1 uses a linearized, outer-approximation approach, while value 0 solves continuous QCP relaxations at each node. The default setting (-1) chooses automatically.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Continuous QCP relaxations at each node</td>
</tr>
<tr>
<td>1</td>
<td>Linearized, outer-approximation approach</td>
</tr>
</tbody>
</table>

**mircuts (integer):** MIR cut generation

See the description of the global Cuts parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**modkcuts (integer):** Mod-k cut generation

...
See the description of the global \texttt{Cuts} parameter for further information.

Default: \texttt{-1}

\textbf{multimipstart} (\textit{string}): Use multiple (partial) mipstarts provided via gdx files

Specifies (multiple) GDX files with values for the variables. Each file is treated as one initial guess for the MIP start. These MIP starts are added in addition to the initial guess provided by the level attribute. A MIP start GDX file can be created, for example, by using the command line option \texttt{savepoint}.

This option further allows to add partial MIP starts where some variable level records are not given. GUROBI will then try to construct a full MIP start out of it. In order to not provide a certain variable level record, simply not include that record in the GDX file or set the level value to NA or UNDEF.

This option requires \texttt{mipstart} enabled.

\textbf{multiobjmethod} (\textit{integer}): Warm-start method to solve for subsequent objectives

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the \texttt{Method} parameter. This parameter determines the algorithm used to solve for subsequent objectives. As with the \texttt{Method} parameters, values of 0 and 1 use primal and dual simplex, respectively. A value of 2 indicates that warm-start information from previous solves should be discarded, and the model should be solved from scratch (using the algorithm indicated by the \texttt{Method} parameter). The default setting of \texttt{-1} usually chooses primal simplex.

Default: \texttt{-1}

\textbf{multiobjpre} (\textit{integer}): Initial presolve on multi-objective models

Controls the initial presolve level used for multi-objective models. Value 0 disables the initial presolve, value 1 applies presolve conservatively, and value 2 applies presolve aggressively. The default \texttt{-1} value usually applies presolve conservatively. Aggressive presolve may increase the chance of the objective values being slightly different than those for other options.

Default: \texttt{-1}

\textbf{multobj} (\textit{boolean}): Controls the hierarchical optimization of multiple objectives

Default: 0

\textbf{names} (\textit{boolean}): Indicator for loading names

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not load GAMS names into Gurobi model</td>
</tr>
<tr>
<td>1</td>
<td>Load GAMS names into Gurobi model</td>
</tr>
</tbody>
</table>

\textbf{networkcuts} (\textit{integer}): Network cut generation

See the description of the global \texttt{Cuts} parameter for further information.

Default: \texttt{-1}
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**nlpheur (boolean):** Controls the NLP heuristic for non-convex quadratic models

The NLP heuristic uses a non-linear barrier solver to find feasible solutions to non-convex quadratic models. It can often find solutions much more quickly than the alternative, but in some cases it can consume significant runtime without producing a solution.

Note: Only affects non-convex quadratic models.

Default: 1

**nodefiledir (string):** Directory for MIP node files

Determines the directory into which nodes are written when node memory usage exceeds the specified NodelfileStart value.

Default: .

**nodefilestart (real):** Memory threshold for writing MIP tree nodes to disk

Controls the point at which MIP tree nodes are written to disk. Whenever node storage exceeds the specified value (in GBytes), nodes are written to disk.

Default: maxdouble

**nodelimit (real):** MIP node limit

Default: maxdouble

**nodemethod (integer):** Method used to solve MIP node relaxations

Algorithm used for MIP node relaxations. Note that barrier is not an option for MIQP node relaxations.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>1</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>2</td>
<td>Barrier</td>
</tr>
</tbody>
</table>

**nonconvex (integer):** Control how to deal with non-convex quadratic programs

Sets the strategy for handling non-convex quadratic objectives or non-convex quadratic constraints. With setting 0, an error is reported if the original user model contains non-convex quadratic constructs. With setting 1, an error is reported if non-convex quadratic constructs could not be discarded or linearized during presolve. With setting 2, non-convex quadratic
problems are solved by means of translating them into bilinear form and applying spatial branching. The default -1 setting is currently equivalent to 1, and may change in future releases to be equivalent to 2.

Default: -1

norelheurtime (real): Limits the amount of time (in seconds) spent in the NoRel heuristic

Limits the amount of time (in seconds) spent in the NoRel heuristic. This heuristic searches for high-quality feasible solutions before solving the root relaxation. It can be quite useful on models where the root relaxation is particularly expensive. Note that this parameter will introduce non-determinism - different runs may take different paths. Use norelheurwork parameter for deterministic results.

Default: 0

norelheurwork (real): Limits the amount of work performed by the NoRel heuristic

Limits the amount of work spent in the NoRel heuristic. This heuristic searches for high-quality feasible solutions before solving the root relaxation. It can be quite useful on models where the root relaxation is particularly expensive. The work metric used in this parameter is tough to define precisely. A single unit corresponds to roughly a second, but this will depend on the machine, the core count, and in some cases the model. You may need to experiment to find a good setting for your model.

Default: 0

normadjust (integer): Simplex pricing norm

Chooses from among multiple pricing norm variants. The default value of -1 chooses automatically.

Default: -1

numericfocus (integer): Set the numerical focus

The NumericFocus parameter controls the degree to which the code attempts to detect and manage numerical issues. The default setting makes an automatic choice, with a slight preference for speed. Settings 1-3 increasingly shift the focus towards being more careful in numerical computations. With higher values, the code will spend more time checking the numerical accuracy of intermediate results, and it will employ more expensive techniques in order to avoid potential numerical issues.

Default: 0

objnabstol (string): Allowable absolute degradation for objective

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is ObjNAbstol ObjVarName value.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value \( z \) when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most ObjNAbstol.

objnreltol (string): Allowable relative degradation for objective
This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is `ObjNRelTol ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value $z$ when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most $ObjNRelTol \times |z|$.

**objscale (real): Objective scaling**

Divides the model objective by the specified value to avoid numerical errors that may result from very large objective coefficients. The default value of 0 decides on the scaling automatically. A value less than zero uses the maximum coefficient to the specified power as the scaling (so `ObjScale=-0.5` would scale by the square root of the largest objective coefficient).

Range: $[-1, \infty]$  
Default: 0

**optimalitytol (real): Dual feasibility tolerance**

Reduced costs must all be larger than `OptimalityTol` in the improving direction in order for a model to be declared optimal.

Range: `[1e-09, 0.01]`  
Default: `1e-06`

**.partition (integer): Variable partition value**

The MIP solver can perform a solution improvement heuristic using user-provided partition information. The provided partition number can be positive, which indicates that the variable should be included when the correspondingly numbered sub-MIP is solved, 0 which indicates that the variable should be included in every sub-MIP, or -1 which indicates that the variable should not be included in any sub-MIP. Variables that are not included in the sub-MIP are fixed to their values in the current incumbent solution.

To give an example, imagine you are solving a model with 400 variables and you set the partition attribute to -1 for variables 0-99, 0 for variables 100-199, 1 for variables 200-299, and 2 for variables 300-399. The heuristic would solve two sub-MIP models: sub-MIP 1 would fix variables 0-99 and 300-399 to their values in the incumbent and solve for the rest, while sub-MIP 2 would fix variables 0-99 and 200-299.

The parameter `PartitionPlace` controls the use of the heuristic. The partition numbers are specified through the option `.partition`. The syntax for `dot options` is explained in the Introduction chapter of the Solver Manual.

Default: 0

**partitionplace (integer): Controls when the partition heuristic runs**

This option works in combination with the `Partition` number for variables. Setting this option and providing some partitions enables the partitioning heuristic, which uses large-neighborhood search to try to improve the current incumbent solution.

This parameter determines where that heuristic runs. Options are:

- Before the root relaxation is solved (16)
• At the start of the root cut loop (8)
• At the end of the root cut loop (4)
• At the nodes of the branch-and-cut search (2)
• When the branch-and-cut search terminates (1)

The parameter value is a bit vector, where each bit turns the heuristic on or off at that place. The numerical values next to the options listed above indicate which bit controls the corresponding option. Thus, for example, to enable the heuristic at the beginning and end of the root cut loop (and nowhere else), you would set the 8 bit and the 4 bit to 1, which would correspond to a parameter value of 12.

The recommended value is 15 which indicates that every option except the first one listed above is enabled.

Default: 15

**perturbvalue** *(real)*: Simplex perturbation magnitude

Range: \([0, \infty]\)

Default: 0.0002

**poolgap** *(real)*: Relative gap for solutions in pool

Determines how large a gap to tolerate in stored solutions. When this parameter is set to a non-default value, solutions whose objective values exceed that of the best known solution by more than the specified (relative) gap are discarded. For example, if the MIP solver has found a solution at objective 100, then a setting of **PoolGap=0.2** would discard solutions with objective worse than 120 (assuming a minimization objective).

Default: maxdouble

**poolgapabs** *(real)*: Absolute gap for solutions in pool

Determines how large a (absolute) gap to tolerate in stored solutions. When this parameter is set to a non-default value, solutions whose objective values exceed that of the best known solution by more than the specified (absolute) gap are discarded. For example, if the MIP solver has found a solution at objective 100, then a setting of **PoolGapAbs=20** would discard solutions with objective worse than 120 (assuming a minimization objective).

Default: maxdouble

**poolsearchmode** *(integer)*: Choose the approach used to find additional solutions

With the default setting (**PoolSearchMode=0**), the MIP solver tries to find an optimal solution to the model. It keeps other solutions found along the way, but those are incidental. By setting this parameter to a non-default value, the MIP search will continue after the optimal solution has been found in order to find additional, high-quality solutions. With a setting of 2, it will find the n best solutions, where n is determined by the value of the **PoolSolutions** parameter. With a setting of 1, it will try to find additional solutions, but with no guarantees about the quality of those solutions. The cost of the solve will increase with increasing values of this parameter.

Once optimization is complete, the **PoolObjBound** attribute (printed to the log) can be used to evaluate the quality of the solutions that were found. For example, a value of **PoolObjBound=100** indicates that there are no other solutions with objective better 100, and thus that any known solutions with objective better than 100 are better than any as-yet undiscovered solutions.

Default: 0
**poolsolutions** *(integer)*: Number of solutions to keep in pool

Determines how many MIP solutions are stored. For the default value of PoolSearchMode, these are just the solutions that are found along the way in the process of exploring the MIP search tree. For other values of PoolSearchMode, this parameter sets a target for how many solutions to find, so larger values will impact performance.

Default: 10

**precrush** *(boolean)*: Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model

Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model. This parameter is turned on when you use BCH with Gurobi.

Default: 0

**predeprrow** *(integer)*: Presolve dependent row reduction

Controls the presolve dependent row reduction, which eliminates linearly dependent constraints from the constraint matrix. The default setting (-1) applies the reduction to continuous models but not to MIP models. Setting 0 turns the reduction off for all models. Setting 1 turns it on for all models.

Default: -1

**predual** *(integer)*: Presolve dualization

Depending on the structure of the model, solving the dual can reduce overall solution time. The default setting uses a heuristic to decide. Setting 0 forbids presolve from forming the dual, while setting 1 forces it to take the dual. Setting 2 employs a more expensive heuristic that forms both the presolved primal and dual models (on two threads), and heuristically chooses one of them.

Default: -1

**premiqcpform** *(integer)*: Format of presolved MIQCP model

Option 0 leaves the model in MIQCP form, so the branch-and-cut algorithm will operate on a model with arbitrary quadratic constraints. Option 1 always transforms the model into MISOCP form; quadratic constraints are transformed into second-order cone constraints. Option 2 always transforms the model into disaggregated MISOCOP form; quadratic constraints are transformed into rotated cone constraints, where each rotated cone contains two terms and involves only three variables.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Always leaves the model in MIQCP form</td>
</tr>
<tr>
<td>1</td>
<td>Always transforms the model into MISOCOP form</td>
</tr>
<tr>
<td>2</td>
<td>Always transforms the model into disaggregated MISOCOP form</td>
</tr>
</tbody>
</table>

**prepasses** *(integer)*: Presolve pass limit

Limits the number of passes performed by presolve. The default setting (-1) chooses the number of passes automatically.
**preqlinearize (integer):** Presolve Q matrix linearization

Options 1 and 2 attempt to linearize quadratic constraints or a quadratic objective, potentially transforming an MIQP or MIQCP model into an MILP. Option 1 focuses on getting a strong LP relaxation. Option 2 aims for a compact relaxation. Option 0 always leaves Q matrices unmodified. The default setting (-1) chooses automatically.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Linearization off</td>
</tr>
<tr>
<td>1</td>
<td>Force Linearization and get strong LP relaxation</td>
</tr>
<tr>
<td>2</td>
<td>Force Linearization and get compact relaxation</td>
</tr>
</tbody>
</table>

**Presolve (integer):** Presolve level

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**presos1bigm (real):** Controls largest coefficient in SOS1 reformulation

Controls the automatic reformulation of SOS1 constraints into binary form. SOS1 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS1 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of -1 chooses a threshold automatically. You should set the parameter to 0 to shut off SOS1 reformulation entirely, or a large value to force reformulation.

Range: [-1, 1e+10]

Default: -1

**presos1encoding (integer):** Controls SOS1 reformulation

Controls the automatic reformulation of SOS1 constraints. Such constraints can be handled directly by the MIP branch-and-cut algorithm, but they are often handled more efficiently by reformulating them using binary or integer variables. There are several different ways to perform this reformulation; they differ in their size and strength. Smaller reformulations add fewer variables and constraints to the model. Stronger reformulations reduce the number of branch-and-cut nodes required to solve the resulting model.

Options 0 and 1 of this parameter encode an SOS1 constraint using a formulation whose size is linear in the number of SOS members. Option 0 uses a so-called multiple choice model. It
usually produces an LP relaxation that is easier to solve. Option 1 uses an incremental model. It often gives a stronger representation, reducing the amount of branching required to solve harder problems.

Options 2 and 3 of this parameter encode the SOS1 using a formulation of logarithmic size. They both only apply when all the variables in the SOS1 are non-negative. Option 3 additionally requires that the sum of the variables in the SOS1 is equal to 1. Logarithmic formulations are often advantageous when the SOS1 constraint has a large number of members. Option 2 focuses on a formulation whose LP relaxation is easier to solve, while option 3 has better branching behaviour.

The default value of -1 chooses a reformulation for each SOS1 constraint automatically. Note that the reformulation of SOS1 constraints is also influenced by the PreSOS1BigM parameter. To shut off the reformulation entirely you should set that parameter to 0.

Default: -1

**presos2bign** *(real)*: Controls largest coefficient in SOS2 reformulation

Controls the automatic reformulation of SOS2 constraints into binary form. SOS2 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS2 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of 0 disables the reformulation. You can set the parameter to -1 to choose an automatic approach, or a large value to force reformulation.

Range: [-1, 1e+10]

Default: -1

**presos2encoding** *(integer)*: Controls SOS2 reformulation

Controls the automatic reformulation of SOS2 constraints. Such constraints can be handled directly by the MIP branch-and-cut algorithm, but they are often handled more efficiently by reformulating them using binary or integer variables. There are several different ways to perform this reformulation; they differ in their size and strength. Smaller reformulations add fewer variables and constraints to the model. Stronger reformulations reduce the number of branch-and-cut nodes required to solve the resulting model.

Options 0 and 1 of this parameter encode an SOS2 constraint using a formulation whose size is linear in the number of SOS members. Option 0 uses a so-called multiple choice model. It usually produces an LP relaxation that is easier to solve. Option 1 uses an incremental model. It often gives a stronger representation, reducing the amount of branching required to solve harder problems.

Options 2 and 3 of this parameter encode the SOS2 using a formulation of logarithmic size. They both only apply when all the variables in the SOS2 are non-negative. Option 3 additionally requires that the sum of the variables in the SOS2 is equal to 1. Logarithmic formulations are often advantageous when the SOS2 constraint has a large number of members. Option 2 focuses on a formulation whose LP relaxation is easier to solve, while option 3 has better branching behaviour.

The default value of -1 chooses a reformulation for each SOS2 constraint automatically. Note that the reformulation of SOS2 constraints is also influenced by the PreSOS2BigM parameter. To shut off the reformulation entirely you should set that parameter to 0.

Default: -1

**presparsify** *(integer)*: Presolve sparsify reduction

This reduction can sometimes significantly reduce the number of nonzero values in the presolved model.

Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Disable the presolve sparsify reduction</td>
</tr>
<tr>
<td>1</td>
<td>Enable the presolve sparsify reduction</td>
</tr>
</tbody>
</table>

**printoptions** *(boolean)*: List values of all options to GAMS listing file

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not list option values to GAMS listing file</td>
</tr>
<tr>
<td>1</td>
<td>List option values to GAMS listing file</td>
</tr>
</tbody>
</table>

**.prior** *(real)*: Branching priorities

GAMS allows to specify priorities for discrete variables only. Gurobi can detect that continuous variables are implied discrete variables and can utilize priorities. Such priorities can be specified through a GAMS/Gurobi solver option file. The syntax for dot options is explained in the Introduction chapter of the Solver Manual. The priorities are only passed on to Gurobi if the model attribute priorOpt is turned on.

Default: 1

**projimpliedcuts** *(integer)*: Projected implied bound cut generation

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Moderate</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**psdcuts** *(integer)*: PSD cut generation

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Moderate</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**psdtol** *(real)*: Positive semi-definite tolerance

Positive semi-definite tolerance (for QP/MIQP). Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to automatically perform on the Q matrix in order to correct minor PSD violations. If a larger perturbation is required, the optimizer will terminate stating the problem is not PSD.

Range: [0, \( \infty \)]

Default: 1e-06
**pumppasses** (*integer*): Feasibility pump heuristic control

Note that this heuristic is only applied at the end of the MIP root.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

Default: -1

**qcpdual** (*boolean*): Compute dual variables for QCP models

Determines whether dual variable values are computed for QCP models. Computing them can add significant time to the optimization, so you should turn this parameter to 0 if you do not need them.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not compute dual for QCP problem</td>
</tr>
<tr>
<td>1</td>
<td>Compute dual for QCP problem</td>
</tr>
</tbody>
</table>

**qextractalg** (*integer*): quadratic extraction algorithm in GAMS interface

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.</td>
</tr>
<tr>
<td>2</td>
<td>DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.</td>
</tr>
</tbody>
</table>

**quad** (*integer*): Quad precision computation in simplex

Enables or disables quad precision computation in simplex. The -1 default setting allows the algorithm to decide.

Default: -1

**readparams** (*string*): Read Gurobi parameter file

**relaxliftcuts** (*integer*): Relax-and-lift cut generation

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>
rerun (integer): Resolve without presolve in case of unbounded or infeasible 

In case Gurobi reports *Model was proven to be either infeasible or unbounded*, this option decides about a resolve without presolve which will determine the exact model status. If the option is set to *auto*, which is the default, and the model fits into demo limits, the problems is resolved.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No</td>
</tr>
<tr>
<td>0</td>
<td>Auto</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

rins (integer): RINS heuristic 

Default value (-1) chooses automatically. A value of 0 shuts off RINS. A positive value \( n \) applies RINS at every \( n \)-th node of the MIP search tree.

Default: -1

rltcuts (integer): RLT cut generation

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

rngrestart (string): Write GAMS readable ranging information file 

If the extension specified is `.gdx`, a GDX file is exported, and a GAMS file otherwise.

scaleflag (integer): Model scaling

Controls model scaling. By default, the rows and columns of the model are scaled in order to improve the numerical properties of the constraint matrix. The scaling is removed before the final solution is returned. Scaling typically reduces solution times, but it may lead to larger constraint violations in the original, unscaled model. Turning off scaling *ScaleFlag=0* can sometimes produce smaller constraint violations. Choosing a more aggressive scaling option *ScaleFlag=2* can sometimes improve performance for particularly numerically difficult models.

Default: -1

seed (integer): Modify the random number seed

Modifies the random number seed. This acts as a small perturbation to the solver, and typically leads to different solution paths.

Default: 0

sensitivity (boolean): Provide sensitivity information

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not provide sensitivity information</td>
</tr>
<tr>
<td>1</td>
<td>Provide sensitivity information</td>
</tr>
</tbody>
</table>

**sifting** (*integer*): Sifting within dual simplex

Enables or disables sifting within dual simplex. Sifting is often useful for LP models where the number of variables is many times larger than the number of constraints. With a *Moderate* setting, sifting will be applied to LP models and to the root node for MIP models. With an *Aggressive* setting, sifting will be also applied to the nodes of a MIP. Note that this parameter has no effect if you aren’t using dual simplex. Note also that sifting will be skipped in cases where it is obviously a worse choice, even when sifting has been selected.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Moderate</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

**siftmethod** (*integer*): LP method used to solve sifting sub-problems

Note that this parameter only has an effect when you are using dual simplex and sifting has been selected (either by the automatic method, or through the *Sifting* parameter).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Primal Simplex</td>
</tr>
<tr>
<td>1</td>
<td>Dual Simplex</td>
</tr>
<tr>
<td>2</td>
<td>Barrier</td>
</tr>
</tbody>
</table>

**simplexpricing** (*integer*): Simplex variable pricing strategy

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Partial Pricing</td>
</tr>
<tr>
<td>1</td>
<td>Steepest Edge</td>
</tr>
<tr>
<td>2</td>
<td>Devex</td>
</tr>
<tr>
<td>3</td>
<td>Quick-Start Steepest Edge</td>
</tr>
</tbody>
</table>

**solfiles** (*string*): Location to store intermediate solution files

During the MIP solution process, multiple incumbent solutions are typically found on the path to finding a proven optimal solution. Setting this parameter to a non-empty string causes these solutions to be written to files (in .sol format) as they are found. The MIP solver will
append \_\_n.sol to the value of the parameter to form the name of the file that contains solution number \_\_n. For example, setting the parameter to value solutions/mymodel will create files mymodel\_0.sol, mymodel\_1.sol, etc., in directory solutions.

solnpooldumpfile (string): Controls export of alternate MIP solutions

The GDX file specified by this option will contain a set call index that contains the names of GDX files with the individual solutions. For details see example model dumpsol in the GAMS Test Library. The option PoolSolutions, PoolSearchModel, and PoolGap control the search for alternative solutions. Please also refer to the section Solution Pool.

solnpoolmerge (string): Controls export of alternate MIP solutions for merged GDX solution file

Similar to SolnPool this option stores multiple alternative solutions to a MIP problem, but in a single GDX file. The GDX file specified by this option will contain all variables with an additional first index (determined through SolnPoolPrefix) as parameters. The option PoolSolutions, PoolSearchModel, and PoolGap control the search for alternative solutions. Please also refer to the section Solution Pool.

solnpoollimitsol (integer): Maximum number of variable symbols when writing merged GDX solution file

Default: 10

solnpoolprefix (string): First dimension of variables for merged GDX solution file or file name prefix for GDX solution files

Default: soln

solutionlimit (integer): MIP feasible solution limit

Default: maxint

solvedumpfile (boolean): Indicator for solving the fixed problem for a MIP to get a dual solution

Default: true

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not solve the fixed problem</td>
</tr>
<tr>
<td>1</td>
<td>Solve the fixed problem</td>
</tr>
</tbody>
</table>

startnodelimit (integer): Node limit for MIP start sub-MIP

This parameter limits the number of branch-and-bound nodes explored when completing a partial MIP start. The default value of -1 uses the value of the SubMIPNodes parameter. A value of -2 means to only check full MIP starts for feasibility and to ignore partial MIP starts. A value of -3 shuts off MIP start processing entirely. Non-negative values are node limits.

Default: -1

strongcgcuts (integer): Strong-CG cut generation

Controls Strong Chvátal-Gomory (Strong-CG) cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: -1
submipcuts (integer): Sub-MIP cut generation

See the description of the global Cuts parameter for further information.

Default: -1

submipnodes (integer): Nodes explored by sub-MIP heuristics

Limits the number of nodes explored by the heuristics, like RINS. Exploring more nodes can produce better solutions, but it generally takes longer.

Default: 500

symmetry (integer): Symmetry detection

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

threads (integer): Number of parallel threads to use

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

Default: GAMS threads

timelimit (real): Time limit

Default: GAMS reslim

tunecleanup (real): Enables a tuning cleanup phase

Enables a cleanup phase at the end of tuning. The parameter indicates the percentage of total tuning time to devote to this phase, with a goal of reducing the number of parameter changes required to achieve the best tuning result.

Default: 0

tunecriterion (integer): Specify tuning criterion

Modifies the tuning criterion for the tuning tool. The primary tuning criterion is always to minimize the runtime required to find a proven optimal solution. However, for MIP models that don't solve to optimality within the specified time limit, a secondary criterion is needed. Set this parameter to 1 to use the optimality gap as the secondary criterion. Choose a value of 2 to use the objective of the best feasible solution found. Choose a value of 3 to use the best objective bound. Choose 0 to ignore the secondary criterion and focus entirely on minimizing the time to find a proven optimal solution. The default value of -1 chooses automatically.

Default: -1

tunejobs (integer): Enables distributed tuning
Enables distributed parallel tuning, which can significantly increase the performance of the tuning tool. A value of $n$ causes the tuning tool to distribute tuning work among $n$ parallel jobs. These jobs are distributed among a set of workers. Use the WorkerPool parameter to provide a list of available workers.

Note that distributed tuning is most effective when the workers have similar performance. Distributed tuning doesn’t attempt to normalize performance by worker, so it can incorrectly attribute a boost in performance to a parameter change when the associated setting is tried on a worker that is significantly faster than the others.

Default: 0

tunemetric (integer): Metric to aggregate results into a single measure

A single tuning run typically produces multiple timing results for each candidate parameter set, either as a result of performing multiple trials, or tuning multiple models, or both. This parameter controls how these results are aggregated into a single measure. The default setting (-1) chooses the aggregation automatically; setting 0 computes the average of all individual results; setting 1 takes the maximum.

Default: -1

tuneoutput (integer): Tuning output level

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output</td>
</tr>
<tr>
<td>1</td>
<td>Summary output only when a new best parameter set is found</td>
</tr>
<tr>
<td>2</td>
<td>Summary output for each parameter set that is tried</td>
</tr>
<tr>
<td>3</td>
<td>Summary output, plus detailed solver output, for each parameter set tried</td>
</tr>
</tbody>
</table>

tuneresults (integer): Number of improved parameter sets returned

The tuning tool often finds multiple parameter sets that produce better results than the baseline settings. This parameter controls how many of these sets should be retained when tuning is complete. The default value retains the best results that were found for each count of changed parameters. In other words, it retains the best result for one changed parameter, for two changed parameter, etc. Results that aren’t on the efficient frontier are discard.

Default: -1

tunetargetmipgap (real): A target gap to be reached

A target gap to be reached. As soon as the tuner has found parameter settings that allow Gurobi to reach the target gap for the given model(s), it stops trying to improve parameter settings further. Instead, the tuner switches into the cleanup phase (see TuneCleanup parameter).

Default: 0

tunetargettime (real): A target runtime in seconds to be reached

A target runtime in seconds to be reached. As soon as the tuner has found parameter settings that allow Gurobi to solve the model(s) within the target runtime, it stops trying to improve parameter settings further. Instead, the tuner switches into the cleanup phase (see TuneCleanup parameter).

Default: 0.005
**tunetimelimit** *(real)*: Time limit for tuning

Limits total tuning runtime (in seconds). The default setting (-1) chooses a time limit automatically.

Default: -1

**tunetrials** *(integer)*: Perform multiple runs on each parameter set to limit the effect of random noise

Performance on a MIP model can sometimes experience significant variations due to random effects. As a result, the tuning tool may return parameter sets that improve on the baseline only due to randomness. This parameter allows you to perform multiple solves for each parameter set, using different Seed values for each, in order to reduce the influence of randomness on the results. The default value of 0 indicates an automatic choice that depends on model characteristics.

Note: Only affects mixed integer programming (MIP) models

Default: 0

**Tuning** *(string)*: Parameter Tuning

Invokes the Gurobi parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Gurobi option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in MPS, REW, LP, RLP, and ILP format created by the WriteProb option. Gurobi will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Gurobi option file name without considering the problem provided by GAMS. The result of such a run is the updated GAMS/Gurobi option file with a tuned set of parameters. In case the option TuneResults is larger than 1, GAMS/Gurobi will create a sequence of GAMS/Gurobi option files. The solver and model status returned to GAMS will be NORMAL COMPLETION and NO SOLUTION. Tuning is incompatible with advanced features like FeasOpt of GAMS/Gurobi.

**usebasis** *(integer)*: Use basis from GAMS

If UseBasis is not specified, GAMS (via option BRatio) decides if the starting basis or a primal/dual solution is given to Gurobi. If UseBasis is explicitly set in an option file then the basis or a primal/dual solution is passed to Gurobi independent of the GAMS option BRatio. Please note, if Gurobi uses a starting basis presolve will be skipped.

Default: GAMS bratio

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No basis</td>
</tr>
<tr>
<td>1</td>
<td>Supply basis if basis is full otherwise provide primal dual solution</td>
</tr>
<tr>
<td>2</td>
<td>Supply basis iff basis is full</td>
</tr>
<tr>
<td>3</td>
<td>Supply primal dual solution</td>
</tr>
</tbody>
</table>

**varbranch** *(integer)*: Branch variable selection strategy

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>Pseudo Reduced Cost Branching</td>
</tr>
<tr>
<td>1</td>
<td>Pseudo Shadow Price Branching</td>
</tr>
<tr>
<td>2</td>
<td>Maximum Infeasibility Branching</td>
</tr>
<tr>
<td>3</td>
<td>Strong Branching</td>
</tr>
</tbody>
</table>

**varhint** *(boolean)*: Guide heuristics and branching through variable hints

If you know that a variable is likely to take a particular value in high quality solutions of a MIP model, you can provide this information as a hint. If **VarHint** option is active, GAMS/Gurobi will pass variable levels rounded to the nearest integer as hints to Gurobi if their level is within **TryInt** of an integer. The closer the level is to the rounded integer the higher your level of confidence in this hint. Internally this is recalculated into a Gurobi variable hint priority:

\[
m = \max(10^{-6}, 1 - |x.l - \lfloor x.l \rfloor|)\]

The Gurobi MIP solver will use these variable hints in a number of different ways. Hints will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree. In general, high quality hints should produce high quality MIP solutions faster. In contrast, low quality hints will lead to some wasted effort, but shouldn’t lead to dramatic performance degradations.

Variables hints and **MIP starts** are similar in concept, but they behave in very different ways. If you specify a MIP start, the Gurobi MIP solver will try to build a single feasible solution from the provided set of variable values. If you know a solution, you should use a MIP start to provide it to the solver. In contrast, variable hints provide guidance to the MIP solver that affects the entire solution process. If you have a general sense of the likely values for variables, you should provide them through variable hints.

Default: 0

**workerpassword** *(string)*: Password for distributed worker cluster

When using a distributed algorithm (the distributed concurrent MIP solver or distributed tuning), this parameter allows you to specify the password for the workers listed in the **WorkerPool** parameter.

**workerpool** *(string)*: Distributed worker cluster

When using a distributed algorithm (distributed MIP, distributed concurrent, or distributed tuning), this parameter allows you to specify a Remote Services cluster that will provide distributed workers. You should also specify the access password for that cluster, if there is one, in the **WorkerPassword** parameter.

You can provide a comma-separated list of machines for added robustness. If the first node in the list is unavailable, the client will attempt to contact the second node, etc.

To give an example, if you have a Remote Services cluster that uses port 61000 on a pair of machines named server1 and server2, you could set WorkerPool to `server1:61000,server2:61000`.

**worklimit** *(real)*: Work limit
Limits the total work expended (in work units).

In contrast to the TimeLimit, work limits are deterministic. This means that on the same hardware and with the same parameter and attribute settings, a work limit will stop the optimization of a given model at the exact same point every time. One work unit corresponds very roughly to one second on a single thread, but this greatly depends on the hardware on which Gurobi is running and the model that is being solved.

Note that optimization may not stop immediately upon hitting the work limit. It will stop when the optimization is next in a deterministic state, and it will then perform the required additional computations of the attributes associated with the terminated optimization. As a result, the Work attribute may be larger than the specified WorkLimit upon completion, and repeating the optimization with a WorkLimit set to the Work attribute of the stopped optimization may result in additional computations and a larger attribute value.

Default: `maxdouble`

writeparams (string): Write Gurobi parameter file

writeprob (string): Save the problem instance

zerohalfcuts (integer): Zero-half cut generation

See the description of the global `Cuts` parameter for further information.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Auto</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Conservative</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive</td>
</tr>
</tbody>
</table>

zeroobjnodes (integer): Zero objective heuristic control

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

Default: -1

5.17.8 Setting up a GAMS/Gurobi-Link license

The GAMS/Gurobi-Link requires two licenses:

- A GAMS license with components GAMS/GUROBI-Link and GAMS BASE (to generate models beyond the demo limits)
A Gurobi license which you need to get directly from Gurobi

An attempt to use the GAMS/Gurobi solver with a GAMS/Gurobi-Link license but without a properly set up Gurobi license will result in a licensing error with a message describing the problem. For example, the following message is sent to the log when attempting to solve a model with dimensions beyond the community limits:

--- Executing GUROBI (Solvelink=2): elapsed 0:00:00.047
Gurobi 32.2.0 rc62c018 Released Aug 26, 2020 WEI x86 64bit/MS Window
Gurobi link license.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)

Starting with GAMS distribution 24.7, even demo sized models require a license from Gurobi. An attempt to solve a demo sized model without a Gurobi license installed results in:

--- Executing GUROBI (Solvelink=2): elapsed 0:00:00.016
Gurobi 32.2.0 rc62c018 Released Aug 26, 2020 WEI x86 64bit/MS Window
*** This solver runs with a demo license. No commercial use.
GAMS/Gurobi demo/community requires a Gurobi license from Gurobi Optimization.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)

To make the GAMS/Gurobi-Link work you do not need to download or install the Gurobi software but only your Gurobi license. GAMS will use it's own Gurobi DLL/shared library, so the Gurobi license has to be valid for the Gurobi version GAMS uses. You can download your Gurobi license from www.gurobi.com. Log in to your Gurobi account and go to Download & Licenses -> Your Gurobi licenses.

Now click on the license you want to download and click on Get License Details. On the license detail page, copy the grbgetkey command on the bottom.

Paste the grbgetkey command to a command/terminal prompt. Your GAMS system directory contains the grbgetkey program, so to make sure it is found you should open the terminal via GAMS Studio or add your GAMS system directory to the path variable. After running the grbgetkey command, you will be asked where you would like to save your Gurobi license.

$ ./grbgetkey [...]
info : grbgetkey version 9.0.3, build v9.0.3rc0
info : Contacting Gurobi key server...
info : Key for license ID [...] was successfully retrieved
info : License expires at the end of the day on 2021-08-31
info : Saving license key...

In which directory would you like to store the Gurobi license key file?
[hit Enter to store it in /opt/gurobi]: /home/nb/gurobilic

info : License [...] written to file /home/nb/gurobilic/gurobi.lic
info : You may have saved the license key to a non-default location
info : You need to set the environment variable GRB_LICENSE_FILE before you can use this license key
info : GRB_LICENSE_FILE=/home/nb/gurobilic/gurobi.lic
Once you have saved your gurobi.lic file, you need to make GAMS/Gurobi aware of that license via environment variable GRB_LICENSE_FILE. The environmentVariables section in the GAMS configuration file (available as of GAMS version 31.1.0) is a convenient way to set the GRB_LICENSE_FILE environment variable. This can either be done by manually opening and editing the gmsconfig.yaml file which can be found in one of the standard locations or via the corresponding GAMS Configuration Editor in GAMS Studio. As a result, the configuration file should contain an entry for environment variable GRB_LICENSE_FILE that points to the Gurobi license, e.g.

```yaml
environmentVariables:
  [...]  
  - GRB_LICENSE_FILE:
    value: /home/nb/gurobilic/gurobi.lic
  [...] 
```

As an alternative you could also set GRB_LICENSE_FILE via the usual OS-specific ways to set environment variables.

To test whether the license setup has been successful, you can solve a model from the GAMS Model library, e.g. [INDUS89], where you should get the following output.

```
$ gamslib indus89
Copy ASCII : indus89.gms
$ gams indus89 lp gurobi
--- Job indus89 Start 09/01/20 12:41:41 32.2.0 rc62c018 LEX-LEG x86 64bit/Linux
[...] 
Gurobi 32.2.0 rc62c018 Released Aug 26, 2020 LEG x86 64bit/Linux 

Gurobi link license.
Gurobi library version 9.0.3
[...]
Iteration  Objective  Primal Inf.  Dual Inf.  Time
0 7.1618935e+31 2.041808e+32 7.161894e+01 0s
2990 1.1487366e+05 0.000000e+00 0.000000e+00 0s
Solved in 2990 iterations and 0.32 seconds
Optimal objective 1.148736556e+05
User-callback calls 3040, time in user-callback 0.00 sec
LP status(2): Model was solved to optimality (subject to tolerances).
--- Reading solution for model wsisn
*** Status: Normal completion
--- Job indus89.gms Stop 09/01/20 12:29:49 elapsed 0:00:00.835
```

Note

To install a Gurobi license key as described above, the grbgetkey program needs to be able to communicate with the Gurobi website. If it fails to work, the grbprobe program which can also be found in your GAMS system directory may be used to retrieve the required hardware information which then needs to be manually submitted to the Gurobi Website (detailed instructions can be found [here](#)).
5.18 Gather-Update-Solve-Scatter (GUSS)

5.18.1 Introduction

The purpose of this chapter is to detail an extension of the GAMS modeling system that allows collections of models (parameterized exogenously by a set of samples or indices) to be described, instantiated, and solved efficiently.

As a specific example, we consider the parametric optimization problem $P(s)$ defined by:

$$\min_{x \in X(s)} f(x; s) \text{ s.t. } g(x; s) \leq 0 \quad (5.16)$$

where $s \in S = \{1, \ldots, K\}$. Note that each scenario $s$ represents a different problem for which the optimization variable is $x$. The form of the constraint set as given above is simply for concreteness; equality constraints and range and bound constraints are trivial extensions of the above framework. Clearly the problems $P(s)$ are interlinked. We intend to show how such problems can be easily specified within GAMS, and detail one type of algorithmic extension that can exploit the nature of the linkage. Other extensions of GAMS allow solves to be executed in parallel or by using grid computing resources. Note that in our description we will use the terms indexed, parameterized, and scenario somewhat interchangeably. An extended version of this chapter containing several examples is available as a paper at http://www.gams.com/modlib/adddocs/gusspaper.pdf.

5.18.2 Design Methodology

One of the most important functions of GAMS is to build a model instance from the collection of equations (i.e. an optimization model defined by the GAMS keyword MODEL) and corresponding data (consisting of the content of GAMS (sub)sets and parameters). Such a model instance is constructed or generated when the GAMS execution system executes a SOLVE statement. The generated model instance is passed to a solver which searches for a solution of this model instance and returns status information, statistics, and a (primal and dual) solution of the model instance. After the solver terminates, GAMS brings the solution back into the GAMS database, i.e. it updates the level (.L) and marginal (.M) fields of variable and equation symbols used in the model instance. Hence, the SOLVE statement can be interpreted as a complex operator against the GAMS database. The model instance generated by a SOLVE statement only lives during the execution of this one statement, and hence has no representation within the GAMS language. Moreover, its structure does fit the relational data model of GAMS. A model instance consists of vectors of bounds and right hand sides, a sparse matrix representation of the Jacobian, a representation of the non-linear expressions that allow the efficient calculation of gradient vectors and Hessian matrices, and so on.

This chapter is concerned with solving collections of models that have similar structure but modified data. As an example, consider a linear program of the form:

$$\min c^T x \text{ s.t. } Ax \geq b, \quad \ell \leq x \leq u.$$ 

The data in this problem is $(A, b, c, \ell, u)$. Omitting some details, the following code could be used within GAMS to solve a collection of such linear programs in which each member of the collection has a different $A$ matrix and lower bound $\ell$:

Set i / ... /, j / ... /;
Parameter A(i,j), b(i);
Variable x(j), z, ...;
Equation e(i), ...;
e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
...
model mymodel /all/;

Set s / s1*s10 /;
Parameter
  A_s(s,i,j) 'Scenario data'
  xlo_s(s,j) 'Scenario lower bound for variable x'
  xl_s(s,j) 'Scenario solution for x.l'
  em_s(s,i) 'Scenario solution for e.m';
Loop(s,
  A(i,j) = A_s(s,i,j);
  x.lo(j) = xlo_s(s,j);
  solve mymodel min z using lp;
  xl_s(s,j) = x.l(j);
  em_s(s,i) = e.m(i);
);

Summarizing, we solve one particular model (mymodel) in a loop over s with an unchanged model rim (i.e. the same individual variables and equations) but with different model data and different bounds for the variables. The change in model data for a subsequent solve statement does not depend on the previous model solutions in the loop.

The purpose of this new Gather-Update-Solve-Scatter (GUSS) manager is to provide syntax at the GAMS modeling level that makes an instance of a problem that provides limited access to treat that instance as an object, and allows the modeler to update portions of it iteratively. Specifically, we provide syntax that gives a list of data changes to an instance, and allows these changes to be applied sequentially to the instance (which is then solved without returning to GAMS). Thus, we can simulate a limited set of actions to be applied to the model instance object and retrieve portions of the solution of these changed instances back in the modeling environment. Such changes can be done to any model type in GAMS, including nonlinear problems and mixed integer models. However, the only changes we allow are to named parameters appearing in the equations and lower and upper bounds used in the model definition.

Thus, in the above example GUSS allows us to replace lines 15-21 by

Set dict / s. scenario. ''
  A. param. A_s
  x. lower. xlo_s
  x. level. xl_s
  e. marginal. em_s /;
solve mymodel min z using lp scenario dict;

The three dimensional set dict (you can freely choose the name of this symbol) contains mapping information between symbols in the model (in the first position) and symbols that supply required update data or store solution information (in the third position), and the type of update/storing (in the second position). An exception to this rule is the tuple with label scenario in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. The remaining tuples in the set dict can be grouped into input and output tuples. Input tuples determine the modifications of the model instance prior to solving, while output tuples determine which part of the solution gets saved away. The following keywords can be used in the second position of the set dict:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>param</td>
<td>Supplies scenario data for a parameter used in the model</td>
</tr>
<tr>
<td></td>
<td>lower</td>
<td>Supplies scenario lower bounds for a variable</td>
</tr>
<tr>
<td></td>
<td>upper</td>
<td>Supplies scenario upper bounds for a variable</td>
</tr>
<tr>
<td></td>
<td>fixed</td>
<td>Supplies scenario fixed bounds for a variable</td>
</tr>
<tr>
<td>Output:</td>
<td>level</td>
<td>Stores the levels of a scenario solution of variable or equation</td>
</tr>
<tr>
<td></td>
<td>marginal</td>
<td>Stores the marginals of a scenario solution of variable or equation</td>
</tr>
</tbody>
</table>
Sets in the model cannot be updated. GUSS works as follows: GAMS generates the model instance for the original data. As with regular `SOLVE` statements, all the model data (e.g. parameter A) needs to be defined at this time. The model instance with the original data is also called the base case. The solution of the base case is reported back to GAMS in the regular way and is accessible via the regular `.L` and `.M` fields after the `SOLVE` statement. After solving the base case, the update data for the first scenario is applied to the model. The tuples with `lower`, `upper`, `fixed` update the bounds of the variables, whereas the tuples with `param` update the parameters in the model.

The scenario index `s` needs to be the first index in the parameters mapped in the `set dict`. The update of the model parameters goes far beyond updating the coefficients of the constraint matrix/objective function or the right hand side of an equation, as one can do with some other systems. GAMS stores all the necessary expressions of the constraints with the model instance, so the change in the constraint matrix coefficient is the result of an expression evaluation. For example, consider a term in the calculation of the cost for shipping a variable amount of goods `x(i,j)` between cities `i` and `j`. The expression for shipping cost is `d(i,j)*f*x(i,j)`, i.e. the distance between the cities times a freight rate `f` times the variable amount of goods. In order to find out the sensitivity of the solution with respect to the freight rate `f`, one can solve the same model with different values for `f`. In a matrix representation of the model one would need to calculate the coefficient of `x(i,j)` which is `d(i,j)*f`, but with GUSS it is sufficient to supply different values for `f` that potentially result in many modified coefficients on the matrix level. GUSS evaluates the shipping cost term and communicates the resulting matrix coefficient to the solver reliably behind the scenes.

After the variable bound and the model parameter updates have been applied and the resulting updates to the model instance data structures (e.g. constraint matrix) has been determined, the modified model instance is passed to the solver. Some solvers (e.g. Cplex, Gurobi, SoPlex, and Xpress) allow modifying a model instance. In these cases GUSS only communicates the changes from the previous model instance to the solver. This reduces the amount of data communicated to the solver and also, in the case of an LP model, allows the solver to restart from an advanced basis and its factorization. In the case of an NLP model, this provides initial values. After the solver determines the solution of a model instance, GUSS stores the part of the solution requested by the output tuples of `dict` to some GAMS parameters and continues with the next scenario. GUSS emphasizes on speed and only works with solvers that allow to communicate the model instance through memory, Hence, the following solvers cannot be used as subsolvers of GUSS: ALPHAECP, BARON, CONVERT, DECISC, DECISM, DICOPT, EXAMINER, GAMSCHK, JAMS, KESTREL, LOGMIP, MILES, MPSGE, NLPEC, PATHNLP, SBB, and XA.

### 5.18.3 GUSS Options

The execution of GUSS can be parameterized using some options. Options are not passed through a solver option file but via another tuple in the `dict` set. The keyword in the second position of this tuple is `opt`. A one dimensional parameter is expected in the first position (or the label `''`). This parameter may contain some of the following labels with values:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptfileInit</td>
<td>Option file number for the first solve (default from GAMS OptFile setting)</td>
</tr>
<tr>
<td>Optfile</td>
<td>Option file number for subsequent solves (default 0)</td>
</tr>
<tr>
<td>LogOption</td>
<td>Determines amount of log output:</td>
</tr>
<tr>
<td></td>
<td>0 - Moderate log (default)</td>
</tr>
<tr>
<td></td>
<td>1 - Minimal log</td>
</tr>
<tr>
<td></td>
<td>2 - Detailed log</td>
</tr>
<tr>
<td>NoHotStart</td>
<td>Disable hot start capability in solver that supports hot starts (default 0)</td>
</tr>
<tr>
<td>NoMatchLimit</td>
<td>Limit of unmatched scenario records (default 0)</td>
</tr>
<tr>
<td>RestartType</td>
<td>Determines restart point for the scenarios</td>
</tr>
<tr>
<td></td>
<td>0 - Restart from last solution (default)</td>
</tr>
<tr>
<td></td>
<td>1 - Restart from solution of base case</td>
</tr>
<tr>
<td></td>
<td>2 - Restart from input point</td>
</tr>
</tbody>
</table>
### Options

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SkipBaseCase</strong></td>
<td>Switch for solving the base case (0 solves the base case)</td>
</tr>
<tr>
<td><strong>ReportLastScen</strong></td>
<td>Switch for reporting the solution of the last scenario rather than solution of the base case (default 0)</td>
</tr>
<tr>
<td><strong>SolveEmpty</strong></td>
<td>Limit of solved empty scenarios, afterwards empty scenarios will be skipped (default 0)</td>
</tr>
<tr>
<td><strong>UpdateType</strong></td>
<td>Scenario update mechanism: 0 - Set everything to zero and apply changes (default) 1 - Reestablish base case and apply changes 2 - Build on top of last scenario and apply changes</td>
</tr>
</tbody>
</table>

For the example model above the **UpdateType** setting would mean:

- **UpdateType=0**: \( \text{loop}(s, A(i,j) = A_s(s,i,j)) \)
- **UpdateType=1**: \( \text{loop}(s, A(i,j) = A_{base}(i,j); A(i,j) = A_s(s,i,j)) \)
- **UpdateType=2**: \( \text{loop}(s, A(i,j) = A_s(s,i,j)) \)

The option **SkipBaseCase=1** allows the user to skip the base case. This means only the scenarios are solved and there is no solution reported back to GAMS in the traditional way. The third position in the opt-tuple can contain a parameter for storing the scenario solution attribute information, e.g. model and solve status, or needs to have the label '' . The labels to store solution status information must be known to GAMS, so one needs to declare a set with such labels. A convenient way to enter these attributes is via `System.GUSSModelAttributes`:

```gams
Set ma 'GUSS Model Attributes' / System.GUSSModelAttributes /; display ma;

---- 1 SET ma GUSS Model Attributes
ModelStat, SolveStat, NumInfes, SumInfes, IterUsd
ResUsd, ObjVal, NodUsd, ObjEst, DomUsd
RObj, MaxInfes, MeanInfes
```

The following example shows how to use some of the GUSS options and the use of a parameter to store some solution status information:

```gams
Set h solution headers / System.GUSSModelAttributes /;
Parameter
  o / SkipBaseCase 1, UpdateType 1, Optfile 1 /
  r_s(s,h) Solution status report;
Set dict / s. scenario. ''
  o. opt. r_s
  a. param. a_s
  x. lower. xlo_s
  x. level. xl_s
  e. marginal. em_s /;
solve mymodel min z using lp scenario dict;
```

Please note that the domain set of the solution status report attributes (here \( h \)) must only contain model attributes known to GUSS. If this domain (unless the domain in *) contains a label unknown to GUSS, a compilation error is triggered.
5.18.4 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

Because GUSS changes all model parameters mentioned in the `dict` set to variables, a linear model can produce some non-linear instructions (e.g. \( d(i,j) \cdot f \cdot x(i,j) \) becomes a non-linear expression since \( f \) becomes a variable in the model instance given to GUSS). This also explains why some models compile without complaint, but if the model is used in the context of GUSS, the compile time check of the model will fail because a parameter that is turned into a variable can no longer be used in that way. For example, suppose the model contains a constraint \( e(i) .. \text{sum}(j A(i,j), ...) \). If \( A(i,j) \) is a parameter in the regular model, the compiler will not complain, but if \( A \) becomes a parameter that shows up in the first position of a `param` tuple in the `dict` set, the GAMS compiler will turn \( A \) into a variable and complain that an endogenous variable cannot be used in a `$`-condition.

The sparsity pattern of a model can be greatly affected by GUSS. In a regular model instance GAMS will only generate and pass on non-zero matrix elements of a constraint \( e(i) .. \text{sum}(j, A(i,j) \cdot x(j)) ... \), so the sparsity of \( A \) determines the sparsity of the generated model instance. GUSS allows to use this constraint with different values for \( A \) hence GUSS cannot exclude any of the pairs \( (i,j) \) and generate a dense matrix. The user can enforce some sparsity by explicitly restricting the \( (i,j) \) pairs: \( e(i) .. \text{sum}(ij(i,j), A(i,j) \cdot x(j)) ... \)

Attention

While GUSS is available for many model types quadratic models require special attention. Linear solvers that have been extended to cover (convex) quadratic models, e.g. Cplex, Gurobi, Mosek, Xpress, do not work properly if the modifying parameter affects the left hand side of linear equations or any modifications of quadratic equations. Bound updates as well as changes of the right hand side of linear constraints are okay (see the example of the quadratic support vector machine below). Unfortunately, detecting if a quadratic model is okay or not for a given solver is at the moment difficult to detect, so use caution with quadratic models in combination with such (linear) solvers.

The actual change of the GAMS language required for the implementation of GUSS is minimal. The only true change is the extension of the `SOLVE` statement with the term `SCENARIO dict`. Existing language elements have been used to store symbol mapping information, options, and model result statistics. Some parts of the GUSS presentation look somewhat unnatural, e.g. since `dict` is a three dimensional `set` the specification the scenario `set` using keyword `scenario` requires a third dummy label `'`. However, this approach gives maximum flexibility for future extension, allows reliable consistency checks at compile and execution time, and allows the user to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a GAMS language level.

5.18.5 Applications

5.18.5.1 Cross Validation in GAMS via GUSS

Cross validation is a statistical/machine learning technique that aims to evaluate the generalizability of a classifier (or other decision) process. It does this by setting aside a portion of the data for testing, and uses the remaining data entries to produce the classifier. The testing data is subsequently used to evaluate how well the classifier works. Cross validation performs this whole process a number of times in order to estimate the true power of the classifier.

Ten-fold cross validation is a special case, where the original data is split into ten pieces, and cross validation is performed using each of these ten pieces as the testing set. Thus, the training process is performed ten times, each of which uses the data obtained by deleting the testing set from the whole dataset. We show below how to carry this out using the Gather-Update-Solve-Scatter (GUSS) facility in GAMS.

A paper with the title "GUSS: Solving Collections of Data Related Models within GAMS" that contains two additional application examples for GUSS is available [here](#).
5.18.5.1.1 GUSS formulation in GAMS  

The following example compares the two formulations for a feature-selection model under cross-validation using data files `a_data.inc` and `b_data.inc`. The actual source code for both of these GAMS formulations is available [here](#).

Original GAMS formulation (without the GAMS/DEA interface):

```gams
$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1' /1*1505/
  b 'set for category 2' /1*957/
  o 'observations' /1*14/
  p 'folds to perform' /1*%num_folds%/ 
  f 'maximum features to select' /1*10/

* Read in the data from the data files
parameter a_data(a, o) / $offlisting
  $include "a_data.inc" $onlisting /;

parameter b_data(b, o) / $offlisting
  $include "b_data.inc" $onlisting /;

set a_test(p, a), b_test(p, b) 'testing sets'
  a_trai(a), b_trai(b) 'training sets';

* Define problem
scalar w_tol /1/
  features /6/;

positive variables a_err(a), sla(a)
  b_err(b), slb(b);

variables c,
  weight(o),
  gamma;

binary variable y(o);

equations w_def1(o),
  w_def2(o),
  y_def,
  c_def,
  a_def(a),
  b_def(b);

w_def1(o).. 
  weight(o) =l= w_tol*y(o);
w_def2(o).. 
```

weight(o) = g = -w_tol*y(o);

y_def..  
   sum(o, y(o)) = e = features;

c_def..  
   c = e = sum(a, a_err(a)) + sum(b, b_err(b));

a_def(a)..  
   -sum(o, a_data(a, o)*weight(o)) + gamma + 1 = l = a_err(a) + sla(a);

b_def(b)..  
   sum(o, b_data(b, o)*weight(o)) - gamma + 1 = l = b_err(b) + slb(b);

model train /all/;

$batinclude gentestset.inc "p,a" "p,b"

set headers 'report' / modelstat, solvestat, objval /;
parameter rep(p,headers);
train.optfile = 0;
option limrow=0, limcol=0, solprint=silent, mip=xpress,
   solvelink=%Solvelink.LoadLibrary%, optcr=0, optca=0;

$echo loadmipsol=1 > xpress.opt

loop(p,
   a_err.up(a) = inf; a_err.up(a)$a_test(p,a) = 0;
   b_err.up(b) = inf; b_err.fx(b)$b_test(p,b) = 0;
   sla.fx(a) = 0; sla.up(a)$a_test(p,a) = inf;
   slb.fx(b) = 0; slb.up(b)$b_test(p,b) = inf;
   solve train using mip minimizing c;
   train.optfile = 1; ! use mipstart for the second run
   rep(p,'modelstat') = train.modelstat;
   rep(p,'solvestat') = train.solvestat;
   rep(p,'objval') = train.objval;
);

display rep;

Options file for the original formulation: xpress.opt

loadmipsol=1

The batinclude file gentestset.inc gives instructions for generating the testing sets. It produces a_test and b_test that detail which equations are left out on solve p.

The actual model is set up to include all the data points in the equations a_def and b_def. To delete the equations that correspond to the test set, we introduce nonnegative slack variables into all the equations. We then set the upper bounds of the slack variables to zero in equations corresponding to the training set, and to infinity in equations corresponding to the testing set. At the same time we fix the error measures a_err and b_err belonging to the testing set by setting their upper bounds to zero. Thus the testing set equations are always satisfiable by choice of the slack variables alone - essentially they are discarded from the model as required. An alternative formulation could "include" the data equations that you need in each scenario, but the update from one scenario to the next in the defining data is much larger.

Cross validation formulated using GUSS: This model essentially mimics what the standard model does, but the implementation of the solver loop behind the scenes is much more efficient, and the consequences are that are clear to see if you execute both model runs. The changes are in the last 40 lines of the GAMS code.
$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1' /1*1505/
b 'set for category 2' /1*957/
o 'observations' /1*14/
p 'folds to perform' /1*%num_folds%/f 'maximum features to select' /1*10/

* Read in the data from the data files
parameter a_data(a, o) /
$offlisting
$include "a_data.inc"
$onlisting
/;

parameter b_data(b, o) /
$offlisting
$include "b_data.inc"
$onlisting
/;

set a_test(p,a), b_test(p,b) 'testing sets'
a_trai(a), b_trai(b) 'training sets';

* Define problem
scalar w_tol /1/
    features /6/;

positive variables a_err(a), sla(a)
    b_err(b), slb(b);

variables c,
    weight(o),
    gamma;

binary variable y(o);

equations w_def1(o),
    w_def2(o),
    y_def,
    c_def,
    a_def(a),
    b_def(b);

w_def1(o).
    weight(o) =l= w_tol*y(o);

w_def2(o).
    weight(o) =g= -w_tol*y(o);

y_def..
    sum(o, y(o)) =e= features;

c_def..
c = \sum(a, a_{err}(a)) + \sum(b, b_{err}(b));

a_{def}(a).
-\sum(o, a_{data}(a, o)\times weight(o)) + \gamma + 1 \leq a_{err}(a) + sla(a);

b_{def}(b).
\sum(o, b_{data}(b, o)\times weight(o)) - \gamma + 1 \leq b_{err}(b) + slb(b);

model train /all/;
train.optfile = 1;

$batinclude gentestset.inc "p,a" "p,b"

parameter wval(p,o), gval(p);

set headers 'report' / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt(*) / SkipBaseCase 1, Optfile 1, LogOption 2 /;

set dict / p. scenario.''
    scopt. opt. scenrep
    a_err. upper. aupper
    b_err. upper. bupper
    sla. upper. afree
    slb. upper. bfree
    weight.level. wval
    gamma. level. gval /

$echo loadmipsol=1 > xpress.opt

Parameter aupper(p,a), bupper(p,b), afree(p,a), bfree(p,b);

aupper(p,a)$(not a_test(p,a)) = inf;
bupper(p,b)$(not b_test(p,b)) = inf;

afree(p,a)$a_test(p,a) = inf;
bfree(p,b)$b_test(p,b) = inf;

option mip=xpress, optcr=0, optca=0;
solve train using mip minimizing c scenario dict;
display scenrep, gval;

Firstly, parameters aupper, bupper, afree and bfree are used to set the bounds on the error and slack variables in the testing set equations respectively. The setting of the upper bounds are governed by the syntax shown in the controlling set dict. Furthermore, the output of the classifier (v, gamma) for each fold of the cross validation uses the dict set to place results into the parameters wval and gval respectively. Finally, the GUSS options are used to guarantee that the subsequent solves are instructed to process solver options (Optfile 1) which instruct the solver to use the previous solution to start the branch-and-cut process (loadmipsol=1).

The complete data and model files for this example are found in (galaxy zip archive). The data and model for a second instance based on the Wisconsin Diagnostic Breast Cancer Database is downloadable as (wdbc zip archive).

5.18.5.1.2 Quadratic Programs

GUSS is not limited to linear programs, but can be used more generally. Simple (indexed) quadratic models can be solved using GUSS. The following example illustrates
the use of GUSS for quadratic programs. In this example, a support vector machine is used to determine
a linear classifier that separates data into two categories. We use the following model:

\[
\begin{align*}
\text{Min}_{w,g,z} & \quad (1/2)\|w\|^2 + Cz^Tz \\
\text{subject to} & \quad D(Aw - g) + z \geq 1 \\
& \quad z \geq 0
\end{align*}
\] (5.17)

Here, \(A\) is a matrix containing the training data (patients by features) and \(D\) is a diagonal matrix with
values +1 or −1 (each denoting one of the two classes). \(C\) is a parameter weighting the importance of
maximizing the margin between the classes \((2/\|w\|_2)\) versus minimizing the misclassification error \((z)\).
The solution \(w\) and \(g\) are used to define a separating hyperplane \(\{x | w^Tx = g\}\) to classify (unseen) data
points.

As given, the standard linear support vector machine is not a slice model per se. It becomes a slice model
under cross-validation training, where it is solved multiple times on different pieces of data. In this case,
only the data \(A\) and \(D\) vary between solves, appropriately fitting the definition of a slice model.

The data for this example comes from the Wisconsin Diagnosis Breast Cancer Database, and is available
here. The data was converted to the GAMS file wdbc.gms, which defines \(A\) and \(D\). The actual
source code for the following GAMS formulation is available here.

The GUSS formulation for quadratic svm:

```
$title Ten-fold cross validation example using GUSS
$eolcom

$setglobal num_folds 10

set p /1*%num_folds%/; ! folds to perform

! Read in data
$include "wdbc.gms"

set test(p,i); ! testing set

! Define problem
parameter C /1/;
positive variables z(i);
variables obj, w(k), gamma, slack(i);

equations obj_def, sep_def(i);

obj_def.. obj =e= 1/2*sum(k, sqr(w(k))) + C*sum(i, z(i));
sep_def(i).. D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) + slack(i) =g= 1;

model train /all/;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset2.inc "p,i"
);

set headers report / modelstat, solvestat, objval /
parameter
  scenrep(p,headers)
  scopt / SkipBaseCase 1, LogOption 2 /
```
Because the problem is quadratic, we must use a quadratic program solver. The variable values for weight and gamma could be saved for later testing using the same method as detailed above for the linear case.

The batinclude file `gentestset2.inc` is very similar to `gentestset.inc` from the earlier cross-validation examples. In `gentestset2.inc`, though, only one set is being dealt with rather than two. The complete source GAMS code for this formulation is available in this zip archive.

### 5.18.5.2 DEA Modeling in GAMS via GUSS

Data Envelopment Analysis (DEA) models can be solved most efficiently in GAMS using the Gather-Update-Solve-Scatter (GUSS) facility. This is the preferred method since release 23.7 of GAMS and hence the GAMS/DEA solver is no longer available.

A paper with the title “GUSS: Solving Collections of Data Related Models within GAMS” that contains two additional application examples for GUSS is available here.

#### 5.18.5.2.1 Introduction

The basic (CCR) DEA model is a collection of models indexed by $k$ and defined by

\[
\begin{align*}
\max_{u,v} & \quad u^T Y_{*,k} \\
\text{subject to} & \quad v^T X_{*,k} = 1 \\
& \quad u^T Y \leq v^T X \\
& \quad u, v \geq 0
\end{align*}
\]

where $X, Y$ are data matrices.

Without using GUSS in GAMS, a model would be defined and solved in a loop over $k$, requiring the model to be generated multiple times with different instances for each value of $k$. GUSS is an alternative (and more efficient) way to define the individual programs and pass them to any underlying GAMS solver. In this way, individual programs are not re-generated, but are instead defined as data modifications of each other. This reduces overall model generation time. Further, previous solutions can be used as starting points in later solves to speed up overall processing time.

Some DEA examples compare the two formulations. The actual source code for both of these formulations is available here.
5.18.5.2.2 DEA Examples  Original GAMS formulation (without GUSS): In all these models the model setup and data are given at the top of the file and the code of interest is in the last 10 or so lines. In this setting we loop over the set k and change the data in the objective function and the first constraint of the model explicitly before each solve. We only output a minimal summary of the solution.

$title Data Envelopment Analysis - DEA
$context

Data Envelopment Analysis (DEA) is a technique for measuring the relative performance of organizational units where presence of multiple inputs and outputs makes comparison difficult.

\[
\text{efficiency} = \frac{\text{weighted sum of output}}{\text{weighted sum of input}}
\]

Find weights that maximize the efficiency for one unit while ensuring that no other units has an efficiency < 1 using these weights. A primal and dual formulation is presented.

Dyson, Thanassoulis, and Boussofiane, A DEA Tutorial.
Warwick Business School

$offtext

Sets  i  'units' / Depot1*Depot20 /
  j  'inputs and outputs' / stock, wages, issues, receipts, reqs /
  ji(j) 'inputs' / stock, wages /
  jo(j) 'outputs' / issues, receipts, reqs /;
alias(i,k);

Table data(i,j)

<table>
<thead>
<tr>
<th></th>
<th>stock</th>
<th>wages</th>
<th>issues</th>
<th>receipts</th>
<th>reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depot1</td>
<td>3</td>
<td>5</td>
<td>40</td>
<td>55</td>
<td>30</td>
</tr>
<tr>
<td>Depot2</td>
<td>2.5</td>
<td>4.5</td>
<td>45</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Depot3</td>
<td>4</td>
<td>6</td>
<td>55</td>
<td>45</td>
<td>30</td>
</tr>
<tr>
<td>Depot4</td>
<td>6</td>
<td>7</td>
<td>48</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Depot5</td>
<td>2.3</td>
<td>3.5</td>
<td>28</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>Depot6</td>
<td>4</td>
<td>6.5</td>
<td>48</td>
<td>20</td>
<td>65</td>
</tr>
<tr>
<td>Depot7</td>
<td>7</td>
<td>10</td>
<td>80</td>
<td>65</td>
<td>57</td>
</tr>
<tr>
<td>Depot8</td>
<td>4.4</td>
<td>6.4</td>
<td>25</td>
<td>48</td>
<td>30</td>
</tr>
<tr>
<td>Depot9</td>
<td>3</td>
<td>5</td>
<td>45</td>
<td>64</td>
<td>42</td>
</tr>
<tr>
<td>Depot10</td>
<td>5</td>
<td>7</td>
<td>70</td>
<td>65</td>
<td>48</td>
</tr>
<tr>
<td>Depot11</td>
<td>5</td>
<td>7</td>
<td>45</td>
<td>65</td>
<td>40</td>
</tr>
<tr>
<td>Depot12</td>
<td>2</td>
<td>4</td>
<td>45</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>Depot13</td>
<td>5</td>
<td>7</td>
<td>65</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>Depot14</td>
<td>4</td>
<td>4</td>
<td>38</td>
<td>18</td>
<td>64</td>
</tr>
<tr>
<td>Depot15</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>Depot16</td>
<td>3</td>
<td>6</td>
<td>38</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Depot17</td>
<td>7</td>
<td>11</td>
<td>68</td>
<td>64</td>
<td>54</td>
</tr>
<tr>
<td>Depot18</td>
<td>4</td>
<td>6</td>
<td>25</td>
<td>38</td>
<td>20</td>
</tr>
<tr>
<td>Depot19</td>
<td>3</td>
<td>4</td>
<td>45</td>
<td>67</td>
<td>32</td>
</tr>
<tr>
<td>Depot20</td>
<td>3</td>
<td>6</td>
<td>57</td>
<td>60</td>
<td>40</td>
</tr>
</tbody>
</table>
;

Parameter slice(j) 'slice of data'

Parameter eff_k(i) 'efficiency report';
Positive variables v(ji) 'input weights'
    u(jo) 'output weights';

Variable eff 'efficiency';

Equations defe 'efficiency definition - weighted output'
    denom 'weighted input'
    lime(i) 'output / input < 1';

defe.. eff =e= sum(jo, u(jo)*slice(jo));
denom.. sum(ji, v(ji)*slice(ji)) =e= 1;
lime(i).. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));

model dea /defe, denom, lime /;

set headers / modelstat, solvestat, objval /;
parameter rep(k,headers) 'solution report summary';
option limrow=0, limcol=0, solprint=silent, solvelink=%Solvelink.LoadLibrary%;
loop(k,
    slice(j) = data(k,j);
    solve dea using lp max eff;
    rep(k,'modelstat') = dea.modelstat;
    rep(k,'solvestat') = dea.solvestat;
    rep(k,'objval') = dea.objval;
);

display rep;

The DEA problem formulated using GUSS: In this setting, the solve statement includes an extra keyword
scenario that points to a new set called dict. The contents of this set are directives to GUSS that state
the scenario index is k, the parameter slice is populated from the parameter data and the values of the
variable eff are stored into the parameter eff_k for each scenario solved. More details follow below.

$title Data Envelopment Analysis - DEA

Sets i 'units' / Depot1*Depot20 /
    j 'inputs and outputs '/ stock, wages, issues, receipts, reqs /
    ji(j) 'inputs' / stock, wages /
    jo(j) 'outputs' / issues, receipts, reqs /;
alias(i,k);

Table data(i,j)
    stock wages issues receipts reqs
Depot1 3 5 40 55 30
Depot2 2.5 4.5 45 50 40
Depot3 4 6 55 45 30
Depot4 6 7 48 20 60
Depot5 2.3 3.5 28 50 25
Depot6 4 6.5 48 20 65
Depot7 7 10 80 65 57
Depot8 4.4 6.4 25 48 30
Depot9 3 5 45 64 42
Depot10 5 7 70 65 48
Parameter slice(j) 'slice of data'
    eff_k(i) 'efficiency report';

Positive variables v(ji) 'input weights'
    u(jo) 'output weights';

Variable eff 'efficiency';

Equations defe 'efficiency definition - weighted output'
    denom 'weighted input'
    lime(i) 'output / input < 1';

defe..    eff =e= sum(jo, u(jo)*slice(jo));
denom..   sum(ji, v(ji)*slice(ji)) =e= 1;
lime(i) .. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));
model dea /defe, denom, lime /

set headers 'report' / modelstat, solvewitness, objval /
parameter scenrep(k,headers) 'solution report summary'
    scopt / SkipBaseCase 1 /
set dict / k .scenario.''  
slice .param. data
    eff .level. eff_k
    scopt .opt. scenrep /

slice(j) = 0; option lp=cplex;
solve dea using lp max eff scenario dict;
display scenrep,eff_k;

In the GUSS version we indicate the collection of models to be solved using the set dict. The first element of dict determines the set to be used for the scenario (collection) index, in this case k. The second element of dict then details that in each scenario k, the parameter slice is instantiated using a slice of the parameter data. Essentially, this corresponds to the GAMS statement:

\[
slice(j) = data(k,j)
\]

Note the scenario index k must appear as the first index of the parameter data. The third element of dict allows the modeler to collect information from each solve and store it into a GAMS parameter. Essentially, the third element of dict corresponds to the GAMS statement:

\[
eff_k(k) = eff.l
\]
that gets executed immediately after the solve of scenario \( k \).

More complex scenario models can also be formulated using GUSS, including multiple equations being updated. This is shown by the dual of the basic DEA model, given by

\[
\begin{align*}
\min_{z, \lambda} & \quad z \\
\text{subject to} & \quad X^* \lambda \leq zX_{*,k} \quad \text{(objective)} \\
& \quad Y^* \lambda \geq Y_{*,k} \quad \text{(slice constraint)} \\
& \quad \lambda \geq 0 \quad \text{(core constraint)} \\
\end{align*}
\]

The next example compares the two formulations for this model. The actual source code for both of these formulations is available [here](#).

**Original GAMS formulation (without GUSS):**

```
$title Data Envelopment Analysis - DEA (traditional)

sets i 'units' / Depot1*Depot20 /
    j 'inputs and outputs' / stock, wages, issues, receipts, reqs /
    ji(j) 'inputs' / stock, wages /
    jo(j) 'outputs' / issues, receipts, reqs /;
alias(k,i);

Table data(i,j)
    stock wages issues receipts reqs
  Depot1 3 5 40 55 30
  Depot2 2.5 4.5 45 50 40
  Depot3 4 6 55 45 30
  Depot4 6 7 48 20 60
  Depot5 2.3 3.5 28 50 25
  Depot6 4 6.5 48 20 65
  Depot7 7 10 80 65 57
  Depot8 4.4 6.4 25 48 30
  Depot9 3 5 45 64 42
  Depot10 5 7 70 65 48
  Depot11 5 7 45 65 40
  Depot12 2 4 45 40 44
  Depot13 5 7 65 25 35
  Depot14 4 4 38 18 64
  Depot15 2 3 20 50 15
  Depot16 3 6 38 20 60
  Depot17 7 11 68 64 54
  Depot18 4 6 25 38 20
  Depot19 3 4 45 67 32
  Depot20 3 6 57 60 40;

parameter slice(j) 'slice of data'
  eff_k(i) 'efficiency report';
Variables z 'efficiency'
    lam(i) 'dual weights';
positive variables lam;
Equations dii(ji) 'input duals'
    dio(jo) 'output dual';
```
* dual model

dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);
dio(jo).. sum(i, lam(i)*data(i,jo)) =g= slice(jo);

model deadc dual with CRS / dii, dio /;

parameter rep 'summary report';
option limrow=0, limcol=0, solprint=silent, lp=cplex,
    solvelink=%Solvelink.LoadLibrary%;

loop(k,
    slice(j) = data(k,j);
    solve deadc using lp minimizing z ;
    rep(k,'modelstat') = deadc.modelstat;
    rep(k,'solvestat') = deadc.modelstat;
    rep(k,'objval') = deadc.objval;
);

display rep;

Dual (CRS) DEA model formulated using GUSS: The key modeling statements occur in the last 10 lines below.

$title Data Envelopment Analysis - DEA (dual,GUSS)

sets i 'units' / Depot1*Depot20 /
    j 'inputs and outputs' / stock, wages, issues, receipts, reqs /
    ji(j) 'inputs' / stock, wages /
    jo(j) 'outputs' / issues, receipts, reqs /
alias(k,i);

Table data(i,j)

<table>
<thead>
<tr>
<th></th>
<th>stock</th>
<th>wages</th>
<th>issues</th>
<th>receipts</th>
<th>reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depot1</td>
<td>3</td>
<td>5</td>
<td>40</td>
<td>55</td>
<td>30</td>
</tr>
<tr>
<td>Depot2</td>
<td>2.5</td>
<td>4.5</td>
<td>45</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Depot3</td>
<td>4</td>
<td>6</td>
<td>55</td>
<td>45</td>
<td>30</td>
</tr>
<tr>
<td>Depot4</td>
<td>6</td>
<td>7</td>
<td>48</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Depot5</td>
<td>2.3</td>
<td>3.5</td>
<td>28</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>Depot6</td>
<td>4</td>
<td>6.5</td>
<td>48</td>
<td>20</td>
<td>65</td>
</tr>
<tr>
<td>Depot7</td>
<td>7</td>
<td>10</td>
<td>80</td>
<td>65</td>
<td>57</td>
</tr>
<tr>
<td>Depot8</td>
<td>4.4</td>
<td>6.4</td>
<td>25</td>
<td>48</td>
<td>30</td>
</tr>
<tr>
<td>Depot9</td>
<td>3</td>
<td>5</td>
<td>45</td>
<td>64</td>
<td>42</td>
</tr>
<tr>
<td>Depot10</td>
<td>5</td>
<td>7</td>
<td>70</td>
<td>65</td>
<td>48</td>
</tr>
<tr>
<td>Depot11</td>
<td>5</td>
<td>7</td>
<td>45</td>
<td>65</td>
<td>40</td>
</tr>
<tr>
<td>Depot12</td>
<td>2</td>
<td>4</td>
<td>45</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>Depot13</td>
<td>5</td>
<td>7</td>
<td>65</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>Depot14</td>
<td>4</td>
<td>4</td>
<td>38</td>
<td>18</td>
<td>64</td>
</tr>
<tr>
<td>Depot15</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>Depot16</td>
<td>3</td>
<td>6</td>
<td>38</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Depot17</td>
<td>7</td>
<td>11</td>
<td>68</td>
<td>64</td>
<td>54</td>
</tr>
<tr>
<td>Depot18</td>
<td>4</td>
<td>6</td>
<td>25</td>
<td>38</td>
<td>20</td>
</tr>
<tr>
<td>Depot19</td>
<td>3</td>
<td>4</td>
<td>45</td>
<td>67</td>
<td>32</td>
</tr>
<tr>
<td>Depot20</td>
<td>3</td>
<td>6</td>
<td>57</td>
<td>60</td>
<td>40</td>
</tr>
</tbody>
</table>
;
parameter slice(j) 'slice of data'
    eff_k(i) 'efficiency report';

Variables z       'efficiency'
    lam(i) 'dual weights';

positive variables lam;

Equations dii(ji) 'input duals'
    dio(jo) 'output dual';

* dual model

dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);
dio(jo).. sum(i, lam(i)*data(i,jo)) =g= slice(jo);

model deadc dual with CRS / dii, dio /;

set headers 'report / modelstat, solvestat, objval /;

parameter scenrep(k,headers) 'solution report summary'
    scopt / SkipBaseCase 1 /;

set dict / k. scenario.''
    scopt. opt. scenrep
    slice. param. data
    z. level. eff_k /;

slice(j) = 0; option lp=cplex;
solve deadc using lp min z scenario dict;
display scenrep,eff_k;

Extensions of these models to formulations with weighted outputs or variable returns to scale are easy to
formulate with the scenario solver within GAMS. This extended model can be downloaded [here].

The **DEA model** in the model library is similar to the extended model, but does not make use of GUSS.

### 5.19 HiGHS

**HiGHS** is an optimization package for solving continuous and mixed-integer linear programming problems
(LPs and MIPs) using simplex, interior-point, and branch-and-cut algorithms. HiGHS is developed by the
Edinburgh Research Group in Optimization.

For more detailed information on the implemented simplex method, we refer to [98].

#### 5.19.1 Usage

The following statement can be used inside your GAMS program to specify using HiGHS

```
Option MIP = HIGHS; { or LP or RMIP }
```

The above statement should appear before the Solve statement. If HiGHS was specified as the default
solver during GAMS installation, the above statement is not necessary.
5.19 HiGHS

5.19.1.1 Specification of HiGHS Options

GAMS/HiGHS supports the GAMS parameters reslim, iterlim, nodlim, optca, optcr, cutoff, and threads.

Options can be specified by a HiGHS options file. A HiGHS options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs).

A small example for a highs.opt file is:

```plaintext
solver = ipm
ipm_optimality_tolerance = 1e-6
run_crossover = false
```

It causes HiGHS to use an interior point solver for an LP solve, increases the optimality tolerance to $10^{-6}$, and turns off crossover to a basis solution.

5.19.2 List of HiGHS Options

In the following, we give a detailed list of all HiGHS options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dual_feasibility_toler</td>
<td>Dual feasibility tolerance Range: $[1e-10, \infty]$</td>
<td>1e-07</td>
</tr>
<tr>
<td>highs_analysis_level</td>
<td>Analysis level in HiGHS Range: ${0, ..., 63}$</td>
<td>0</td>
</tr>
<tr>
<td>highs_debug_level</td>
<td>Debugging level in HiGHS Range: ${0, ..., 3}$</td>
<td>0</td>
</tr>
<tr>
<td>infinite_bound</td>
<td>Limit on $</td>
<td>\text{constraint bound}</td>
</tr>
<tr>
<td>infinite_cost</td>
<td>Limit on cost coefficient: values larger than this will be treated as infinite Range: $[1e+15, \infty]$</td>
<td>1e+20</td>
</tr>
<tr>
<td>ipm_iteration_limit</td>
<td>Iteration limit for IPM solver Range: ${0, ..., \infty}$</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>ipm_optimality_tolerance</td>
<td>IPM optimality tolerance Range: $[1e-12, \infty]$</td>
<td>1e-08</td>
</tr>
<tr>
<td>large_matrix_value</td>
<td>Upper limit on $</td>
<td>\text{matrix entries}</td>
</tr>
<tr>
<td>mip_abs_gap</td>
<td>tolerance on absolute gap of MIP, $</td>
<td>\text{ub-lb}</td>
</tr>
<tr>
<td>mip_detect_symmetry</td>
<td>Whether symmetry should be detected Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mip_feasibility_tolerance</td>
<td>MIP feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-10, ∞]</td>
<td></td>
</tr>
<tr>
<td>mip_heuristic_effort</td>
<td>effort spent for MIP heuristics</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>mip_lp_age_limit</td>
<td>maximal age of dynamic LP rows before they are removed from the LP relaxation</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 32767}</td>
<td></td>
</tr>
<tr>
<td>mip_max_improving_sols</td>
<td>limit on the number of improving solutions found to stop the MIP solver prematurely</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_max_leaves</td>
<td>MIP solver max number of leave nodes</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_max_nodes</td>
<td>MIP solver max number of nodes</td>
<td>GAMS nodlim, if &gt; 0, ∞ otherwise</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_max_stall_nodes</td>
<td>MIP solver max number of nodes where estimate is above cutoff bound</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_min_cliquetable_entries_for_parallelism</td>
<td>minimal number of entries in the cliquetable before neighborhood queries of the conflict graph use parallel processing</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_pool_age_limit</td>
<td>maximal age of rows in the cutpool before they are deleted</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 1000}</td>
<td></td>
</tr>
<tr>
<td>mip_pool_soft_limit</td>
<td>soft limit on the number of rows in the cutpool for dynamic age adjustment</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_pscost_minreliable</td>
<td>minimal number of observations before pseudo costs are considered reliable</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>mip_rel_gap</td>
<td>tolerance on relative gap,</td>
<td>ub-(lb</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>mip_report_level</td>
<td>MIP solver reporting level</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>objective_bound</td>
<td>Objective bound for termination</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>objective_target</td>
<td>Objective target for termination</td>
<td>-∞</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>output_flag</td>
<td>Enables or disables solver output</td>
<td>0, if GAMS logoption = 0, otherwise 1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td>choose</td>
</tr>
<tr>
<td>parallel</td>
<td>Parallel option: &quot;off&quot;, &quot;choose&quot; or &quot;on&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>presolve</td>
<td>Presolve option: &quot;off&quot;, &quot;choose&quot; or &quot;on&quot;</td>
<td>choose</td>
</tr>
<tr>
<td>primal_feasibility_tolerance</td>
<td>Primal feasibility tolerance</td>
<td>1e-07</td>
</tr>
<tr>
<td>random_seed</td>
<td>random seed used in HiGHS</td>
<td>0</td>
</tr>
<tr>
<td>sensitivity</td>
<td>Whether to run sensitivity analysis after solving an LP with a simplex method</td>
<td>0</td>
</tr>
<tr>
<td>simplex_crash_strategy</td>
<td>Strategy for simplex crash: off / LTSSF / Bixby (0/1/2)</td>
<td>0</td>
</tr>
<tr>
<td>simplex_dual_edge_weight_strategy</td>
<td>Strategy for simplex dual edge weights: Choose / Dantzig / Devex / Steepest Edge (-1/0/1/2)</td>
<td>-1</td>
</tr>
<tr>
<td>simplex_iteration_limit</td>
<td>Iteration limit for simplex solver</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>simplex_max_concurrency</td>
<td>Maximum level of concurrency in parallel simplex</td>
<td>8</td>
</tr>
<tr>
<td>simplex_min_concurrency</td>
<td>Minimum level of concurrency in parallel simplex</td>
<td>1</td>
</tr>
<tr>
<td>simplex_primal_edge_weight_strategy</td>
<td>Strategy for simplex primal edge weights: Choose / Dantzig / Devex / Steepest Edge (-1/0/1/2)</td>
<td>-1</td>
</tr>
<tr>
<td>simplex_scale_strategy</td>
<td>Simplex scaling strategy: off / choose / equilibration / forced equilibration / max value 0 / max value 1 (0/1/2/3/4/5)</td>
<td>1</td>
</tr>
<tr>
<td>simplex_strategy</td>
<td>Strategy for simplex solver 0 =&gt; Choose; 1 =&gt; Dual (serial); 2 =&gt; Dual (PAMI); 3 =&gt; Dual (SIP); 4 =&gt; Primal</td>
<td>1</td>
</tr>
<tr>
<td>simplex_update_limit</td>
<td>Limit on the number of simplex UPDATE operations</td>
<td>5000</td>
</tr>
<tr>
<td>small_matrix_value</td>
<td>Lower limit on</td>
<td>matrix entries</td>
</tr>
<tr>
<td>solution_file</td>
<td>Solution file</td>
<td>&lt;inputname&gt;.sol</td>
</tr>
<tr>
<td>solver</td>
<td>Solver option: &quot;simplex&quot;, &quot;choose&quot; or &quot;ipm&quot;, ignored for MIP</td>
<td>choose</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>threads</td>
<td>number of threads used by HiGHS (0: automatic)</td>
<td>GAMS threads</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>time_limit</td>
<td>Time limit (seconds)</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>write_model_file</td>
<td>Write model file</td>
<td>&lt;inputname&gt;.lp</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>write_model_to_file</td>
<td>Write the model to a file</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>write_solution_style</td>
<td>Write the solution in style:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0=&gt;Raw (computer-readable); 1=&gt;Pretty (human-readable); 2=&gt;OldRaw (deprecated)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>write_solution_to_file</td>
<td>Write the primal and dual solution to a file</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>allow_un bounded or</td>
<td>whether to spend extra effort to distinguish unboundedness and infeasibility</td>
<td>0</td>
</tr>
<tr>
<td>infeasible</td>
<td>if necessary</td>
<td>boolean</td>
</tr>
<tr>
<td>allowed_cost_scale_factor</td>
<td>Largest power-of-two factor permitted when scaling the costs</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>allowed_matrix_scale_factor</td>
<td>Largest power-of-two factor permitted when scaling the constraint matrix</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>cost_scale_factor</td>
<td>Scaling factor for costs</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([-20, \infty])</td>
<td></td>
</tr>
<tr>
<td>dual_simplex_cost</td>
<td>Dual simplex cost perturbation multiplier: 0 =&gt; no perturbation</td>
<td>1</td>
</tr>
<tr>
<td>perturbation_multiplier</td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>dual_simplex_pivot</td>
<td>Dual simplex pivot growth tolerance</td>
<td>1e-09</td>
</tr>
<tr>
<td>growth_tolerance</td>
<td>Range: ([1, \infty])</td>
<td></td>
</tr>
<tr>
<td>dual_steepest_edge</td>
<td>Tolerance on dual steepest edge weight errors</td>
<td>(\infty)</td>
</tr>
<tr>
<td>weight_error_tolerance</td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>dual_steepest_edge</td>
<td>Threshold on dual steepest edge weight errors for Devex switch</td>
<td>10</td>
</tr>
<tr>
<td>weight_log_error_threshold</td>
<td>Range: ([1, \infty])</td>
<td></td>
</tr>
<tr>
<td>factor_pivot_threshold</td>
<td>Matrix factorization pivot threshold</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0.0008, 0.5])</td>
<td></td>
</tr>
<tr>
<td>factor_pivot_tolerance</td>
<td>Matrix factorization pivot tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>less_infeasible_DSE_check</td>
<td>Check whether LP is candidate for LiDSE</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>less_infeasible_DSE_choose_row</td>
<td>Use LiDSE if LP has right properties</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>log_dev_level</td>
<td>Output development messages: 0 =&gt; none; 1 =&gt; info; 2 =&gt; verbose</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 3}</td>
<td></td>
</tr>
<tr>
<td>lp_presolve_requires_basis_postsolve</td>
<td>Prevents LP presolve steps for which postsolve cannot maintain a basis</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>max_dual_simplex_cleanup_level</td>
<td>Max level of dual simplex cleanup</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>max_dual_simplex_phase1_cleanup_level</td>
<td>Max level of dual simplex phase 1 cleanup</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>no_unnecessary_rebuild_refactor</td>
<td>No unnecessary refactorization on simplex rebuild</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>presolve_pivot_threshold</td>
<td>Matrix factorization pivot threshold for substitutions in presolve</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0.0008, 0.5]</td>
<td></td>
</tr>
<tr>
<td>presolve_substitution_maxfillin</td>
<td>Maximal fillin allowed for substitutions in presolve</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>primal_simplex_bound_perturbation_multiplier</td>
<td>Primal simplex bound perturbation multiplier: 0 =&gt; no perturbation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>rebuild_refactor_solution_error_tolerance</td>
<td>Tolerance on solution error when considering refactorization on simplex rebuild</td>
<td>1e-08</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>run_crossover</td>
<td>Run the crossover routine for IPX</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>simplex_dualise_strategy</td>
<td>Strategy for dualising before simplex</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 1}</td>
<td></td>
</tr>
<tr>
<td>simplex_initial_condition_check</td>
<td>Perform initial basis condition check in simplex</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>simplex_initial_condition_tolerance</td>
<td>Tolerance on initial basis condition in simplex</td>
<td>1e+14</td>
</tr>
<tr>
<td></td>
<td>Range: [1, \infty]</td>
<td></td>
</tr>
<tr>
<td>simplex_permute_strategy</td>
<td>Strategy for permuting before simplex</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 1}</td>
<td></td>
</tr>
<tr>
<td>simplex_price_strategy</td>
<td>Strategy for PRICE in simplex</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 3}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.20 IPOPT and IPOPTH

**COIN-OR IPOPT** (Interior Point Optimizer) is an open-source solver for large-scale nonlinear programming (NLP). The code has been written primarily by Andreas Wächter.

IPOPT implements an interior point line search filter method for nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. For more information on the algorithm we refer to [144] [196] [195] [197] [194] and the [IPOPT documentation]. Most of the IPOPT documentation in the section was taken from the IPOPT manual [104].

#### 5.20.1 Available linear solvers

The performance and robustness of IPOPT on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems.

GAMS/IPOPT includes the sparse solver [MUMPS] [8] [9] (currently the default), and MKL PARDISO [166] [167]. In the commercially licensed GAMS/IPOPTH version, also the Harwell Subroutine Library (HSL) solvers MA27, MA57, HSL_MA86, and HSL_MA97 are available and MA27 is used by default.

MUMPS, MA57, HSL_MA86, and HSL_MA97 use METIS for matrix ordering [103], see also the [METIS manual]. METIS is copyrighted by the regents of the University of Minnesota.

IPOPT and IPOPTH can exploit parallelization of the linear solvers MKL Pardiso, HSL MA86, and HSL MA97 and the linear algebra routines (see next section).

The linear solver is chosen by the `linear_solver` option. Benchmarks have shown that MA57 and HSL_MA97 are often able to outperform MA27 on larger instances. Further, PARDISO often allows for performance that is better than MUMPS and similar to the HSL solvers. If IPOPT fails to solve an instance with PARDISO, it’s worth to try changing the options `pardisomkl_order` and `pardisomkl_max_iterative_refinement_steps`.

It is also possible to use the linear solver PARDISO from the [PARDISO Solver Project] or the [HSL routines] with GAMS/IPOPT if a user provides libraries that can be loaded at runtime. PARDISO from the PARDISO Solver Project can provide performance that exceeds the one of PARDISO from Intel MKL. To build the HSL routines, COIN-OR project ThirdParty-HSL may be useful. See also options `linear_solver`, `linear_system_scaling`, `nlp_scaling_method`, `pardisolib`, and `hsllib`. Note that it is your responsibility to ensure that you are entitled to download and use these routines!
5.20.2 The linear algebra library

The IPOPT library distributed by GAMS and most of the linear solvers used by IPOPT use the Intel oneAPI Math Kernel Library (MKL), which provides a fast and parallel implementation of linear algebra routines (BLAS/LAPACK) and the linear solver PARDISO. MKL chooses an internal code path that provides the best possible performance for the used CPU type. As a consequence, results can be different when changing from one CPU to another. By setting an environment variable, the code path to use can be set by the user. See the Intel MKL documentation regarding Conditional Numerical Reproducibility for more details.

Intel MKL has been optimized for use with Intel CPUs. On CPUs from other vendors, MKL may not use an internal code path that could provide a better performance on that CPU. For example, it may not use AVX2 instructions on an AMD CPU that provides AVX2.

To gain more insight into the use of MKL in GAMS/IPOPT, one may set the environment variable MKL_VERBOSE to 1. This will print out information about the MKL library used, functions being called, time spend there, etc.

5.20.3 Usage

The following statement can be used inside your GAMS program to specify using IPOPT

```
Option NLP = IPOPT;  { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

The above statement should appear before the Solve statement. If IPOPT was specified as the default solver during GAMS installation, the above statement is not necessary.

To use IPOPTH, the statement should be

```
Option NLP = IPOPTH;  { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

5.20.3.1 Specification of Options

IPOPT has many options that can be adjusted for the algorithm (see Section List of IPOPT Options). Options are all identified by a string name, and their values can be of one of three types: Number (real), Integer, or String. Number options are used for things like tolerances, integer options are used for things like maximum number of iterations, and string options are used for setting algorithm details, like the NLP scaling method. Options can be set by creating a ipopt.opt file in the directory you are executing IPOPT.

The ipopt.opt file is read line by line and each line should contain the option name, followed by whitespace, and then the value. Comments can be included with the # symbol. For example, the following is a valid ipopt.opt file:

```
# This is a comment

# Turn off the NLP scaling
nlp_scaling_method none

# Change the initial barrier parameter
mu_init 1e-2

# Set the max number of iterations
max_iter 500
```

GAMS/IPOPT understands currently the following GAMS parameters: reslim (time limit), iterlim (iteration limit), domlim (domain violation limit). Further the option threads can be used to control the number of threads used in the linear algebra routines and the linear solver.
5.20.3.2 Warmstarting IPOPT

As an interior point solver, it is difficult to warm start IPOPT. By default, only the level values of the variables are passed as starting point to IPOPT. Setting the IPOPT option `warm_start_init_point` to `yes` enables that also dual values for variables and constraints are passed to IPOPT.

However, the expected behavior that IPOPT finishes within one iteration if optimal primal and dual values are passed is not reached this way, yet. This is, because IPOPT by default moves any initial value that is close to a bound into the interior. The amount on how much the initial point is moved can be controlled by various `bound_push` and `bound_frac` options. To make IPOPT accept an optimal primal/dual solution within one iteration, it should be sufficient to set the following options:

```
warm_start_init_point       yes
warm_start_bound_push       1e-9
warm_start_bound_frac       1e-9
warm_start_slack_bound_push 1e-9
warm_start_slack_bound_frac 1e-9
warm_start_mult_bound_push  1e-9
```

Further, it can be useful to specify that IPOPT can use a less central path in its first iterations by reducing the value of option `mu_init`. This option is only used if option `mu_strategy` is set to "monotone", so the option file entries would be

```
mu_strategy monotone
mu_init 0.0001
```

Finally, IPOPT by default checks whether it should scale the problem. The computed scaling depends on the starting point, which can be undesired when warmstarting. Thus, it may be useful to turn off scaling via option `nlp_scaling_method`:

```
nlp_scaling_method none
```

If a modified but structurally equivalent problem instance is solved, e.g., via GUSS, option `warm_start_init_point` is automatically set to "yes" for every solve following the first one. If this is not desired, `warm_start_init_point` should explicitly be set to "no" in an IPOPT options file.

5.20.4 Output

This section describes the standard IPOPT console output. The output is designed to provide a quick summary of each iteration as IPOPT solves the problem.

Before IPOPT starts to solve the problem, it displays the problem statistics (number of nonzero-elements in the matrices, number of variables, etc.). Note that if you have fixed variables (both upper and lower bounds are equal), IPOPT may remove these variables from the problem internally and not include them in the problem statistics.

Following the problem statistics, IPOPT will begin to solve the problem and you will see output resembling the following,

```
iter    objective   inf_pr   inf_du  lg(mu)    ||d||    lg(rg)   alpha_du  alpha_pr  ls
0       1.6109693e+01 1.12e+01 5.28e-01 0.00 0.000e+00 - 0.00e+00 0.00e+00 0
1       1.8029749e+01 9.90e-01 6.62e+00 0.10 2.05e+00 - 2.14e-01 1.00e+00 1
2       1.8719906e+01 1.25e-02 9.04e+00 -2.2 5.94e-02 2.0 8.04e-01 1.00e+00 1
```
and the columns of output are defined as

**item**

The current iteration count. This includes regular iterations and iterations while in restoration phase. If the algorithm is in the restoration phase, the letter \( r \) will be appended to the iteration number.

**objective**

The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.

**inf\_pr**

The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraint violation. During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option `inf_pr_output` can be used to switch to the printing of a different quantity. During the restoration phase, this value is the primal infeasibility of the original problem at the current point.

**inf\_du**

The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility (Eq. (4a) in [197]), including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.

**lg(mu)**

\( \log_{10} \) of the value of the barrier parameter \( \mu \).

**\|d\|**

The infinity norm (max) of the primal step (for the original variables \( x \) and the internal slack variables \( s \)). During the restoration phase, this value includes the values of additional variables, \( p \) and \( n \) in Eq. (10) of [197].

**lg(rg)**

\( \log_{10} \) of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (\( \delta_w \) in Eq. (26) of [197]). A dash (\(-\)) indicates that no regularization was done.

**alpha\_du**

The stepsize for the dual variables (\( \alpha^z_k \) in Eq. (14c) of [197]).

**alpha\_pr**

The stepsize for the primal variables (\( \alpha_k \) in Eq. (14a) of [197]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:

- \( f \): f-type iteration in the filter method w/o second order correction
- \( F \): f-type iteration in the filter method w/ second order correction
- \( h \): h-type iteration in the filter method w/o second order correction
- \( H \): h-type iteration in the filter method w/ second order correction
- \( k \): penalty value unchanged in merit function method w/o second order correction
• \( K \): penalty value unchanged in merit function method w/ second order correction
• \( n \): penalty value updated in merit function method w/o second order correction
• \( N \): penalty value updated in merit function method w/ second order correction
• \( R \): Restoration phase just started
• \( w \): in watchdog procedure
• \( s \): step accepted in soft restoration phase
• \( t/T \): tiny step accepted without line search
• \( r \): some previous iterate restored

\[ \text{ls} \]

The number of backtracking line search steps (does not include second-order correction steps).

Note that the step acceptance mechanisms in IPOPT consider the barrier objective function (Eq. (3a) in [197]) which is usually different from the value reported in the \text{objective} column. Similarly, for the purposes of the step acceptance, the constraint violation is measured for the internal problem formulation, which includes slack variables for inequality constraints and potentially scaling of the constraint functions. This value, too, is usually different from the value reported in \text{infpr}. As a consequence, a new iterate might have worse values both for the objective function and the constraint violation as reported in the iteration output, seemingly contradicting globalization procedure.

When the algorithm terminates, IPOPT will output a message to the screen. The following is a list of the possible output messages and a brief description.

**Optimal Solution Found.**

This message indicates that IPOPT found a (locally) optimal point within the desired tolerances.

**Solved To Acceptable Level.**

This indicates that the algorithm did not converge to the "desired" tolerances, but that it was able to obtain a point satisfying the "acceptable" tolerance level as specified by \text{acceptable-\star} options. This may happen if the desired tolerances are too small for the current problem.

**Feasible point for square problem found.**

This message is printed if the problem is "square" (i.e., it has as many equality constraints as free variables) and IPOPT found a point that is feasible w.r.t. \text{constr\_viol\_tol}. It may, however, not be feasible w.r.t. \text{tol}.

**Converged to a point of local infeasibility. Problem may be infeasible.**

The restoration phase converged to a point that is a minimizer for the constraint violation (in the \( \ell_1 \)-norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.

**Search Direction is becoming Too Small.**

This indicates that IPOPT is calculating very small step sizes and making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current scaling.

**Iterates divering; problem might be unbounded.**
This message is printed if the max-norm of the iterates becomes larger than the value of the option `diverging_iterates_tol`. This can happen if the problem is unbounded below and the iterates are diverging.

Stopping optimization at current point as requested by user.

This message is printed if either an interrupt signal was received (e.g., Ctrl+C was pressed) or the domain violation limit is reached.

Maximum Number of Iterations Exceeded.

This indicates that IPOPT has exceeded the maximum number of iterations as specified by the IPOPT option `max_iter` or the GAMS option `iterlim`.

Maximum wallclock time exceeded.

This indicates that IPOPT has exceeded the maximum number of wallclock seconds as specified by the IPOPT option `max_wall_time` or the GAMS option `reslim`.

Maximum CPU time exceeded.

This indicates that IPOPT has exceeded the maximum number of CPU seconds as specified by the IPOPT option `max_cpu_time`.

Restoration Failed!

This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate, does not satisfy the constraint qualification, or if an external or extrinsic function in GAMS provides incorrect derivative information.

Error in step computation!

This messages is printed if IPOPT is unable to compute a step towards a new iterate and the current iterate is not acceptable for the specified tolerances.

A possible reason is that a search direction could not be computed despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large.

Another reason is that the feasibility restoration phase could not be activated because the current iterate is not infeasible. Reasons for this again include that the problem is highly degenerate, badly scaled, or does not satisfy the constraint qualification. Before IPOPT 3.14, this resulted in a Restoration_Failed status code with message ”Restoration phase is called at almost feasible point...”.

Problem has too few degrees of freedom.

This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables (IPOPT removes fixed variables by default, see also the option `fixed_variable_treatment`).

Not enough memory.

An error occurred while trying to allocate memory. The problem may be too large for your current memory and swap configuration. Sometimes it can help to choose a different linear solver.

INTERNAL ERROR: Unknown SolverReturn value - Notify IPOPT Authors.

An unknown internal error has occurred. Please notify the authors of the GAMS/IPOPT link or IPOPT (refer to support@gams.com).
5.20.4.1 Diagnostic Tags for IPOPT

To print additional diagnostic tags for each iteration of IPOPT, set the options print_info_string to yes. With this, a tag will appear at the end of an iteration line with the following diagnostic meaning that are useful to flag difficulties for a particular IPOPT run. The following is a list of possible strings:

- !: Tighten resto tolerance if only slightly infeasible, see Sec. 3.3 in [197]
- A: Current iteration is acceptable (alternate termination)
- a: Perturbation for PD Singularity can't be done, assume singular, see Sec. 3.1 in [197]
- C: Second Order Correction taken, see Sec. 2.4 in [197]
- Dh: Hessian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- Dhj: Hessian/Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- Dj: Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- δx: δx perturbation too large, see Sec. 3.1 in [197]
- e: Cutting back α due to evaluation error (in backtracking line search)
- F−: Filter should be reset, but maximal resets exceeded, see Sec. 2.3 in [197]
- F+: Resetting filter due to last few rejections of filter, see Sec. 2.3 in [197]
- L: Degenerate Jacobian, δc already perturbed, see Sec. 3.1 in [197]
- l: Degenerate Jacobian, δc perturbed, see Sec. 3.1 in [197]
- M: Magic step taken for slack variables (in backtracking line search)
- Nh: Hessian not yet degenerate, see Sec. 3.1 in [197]
- Nhj: Hessian/Jacobian not yet degenerate, see Sec. 3.1 in [197]
- Nj: Jacobian not yet degenerate, see Sec. 3.1 in [197]
- NW: Warm start initialization failed (in Warm Start Initialization)
- q: PD system possibly singular, attempt to improve solution quality, see Sec. 3.1 in [197]
- R: Solution of restoration phase, see Sec. 3.3 in [197]
- s: PD system singular, see Sec. 3.1 in [197]
- S: PD system possibly singular, accept current solution, see Sec. 3.1 in [197]
- T: Square Problem. Set multipliers to zero (default initialization routine)
- Tmax: Trial θ is larger than θmax (filter parameter, Eq. (21) in [197])
- W: Watchdog line search procedure successful, see Sec. 3.2 in [197]
- w: Watchdog line search procedure unsuccessful, stopped, see Sec. 3.2 in [197]
- Wb: Undoing most recent SR1 update, see Sec. 5.4.1 in [23]
- We: Skip Limited-Memory Update in restoration phase, see Sec. 5.4.1 in [23]
- Wp: Safeguard B0 = σI for Limited-Memory Update, see Sec. 5.4.1 in [23]
- Wr: Resetting Limited-Memory Update, see Sec. 5.4.1 in [23]
- Ws: Skip Limited-Memory Update since sTy is not positive, see Sec. 5.4.1 in [23]
- WS: Skip Limited-Memory Update since Δx is too small, see Sec. 5.4.1 in [23]
- y: Dual infeasibility, use least square multiplier update (during IPOPT algorithm)
- z: Apply correction to bound multiplier if too large (during IPOPT algorithm)
## 5.20.5 List of IPOPT Options

### 5.20.5.1 Termination

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptable_compl_inf_tol</td>
<td>&quot;Acceptance&quot; threshold for the complementarity conditions.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_constr_viol_tol</td>
<td>&quot;Acceptance&quot; threshold for the constraint violation.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_dual_inf_tol</td>
<td>&quot;Acceptance&quot; threshold for the dual infeasibility.</td>
<td>1e+10</td>
</tr>
<tr>
<td>acceptable_iter</td>
<td>Number of &quot;acceptable&quot; iterates before triggering termination.</td>
<td>0</td>
</tr>
<tr>
<td>acceptable_obj_change_tol</td>
<td>&quot;Acceptance&quot; stopping criterion based on objective function change.</td>
<td>1e+20</td>
</tr>
<tr>
<td>compl_inf_tol</td>
<td>Desired threshold for the complementarity conditions.</td>
<td>0.0001</td>
</tr>
<tr>
<td>constr_viol_tol</td>
<td>Desired threshold for the constraint and variable bound violation.</td>
<td>1e-06</td>
</tr>
<tr>
<td>diverging_iterates_tol</td>
<td>Threshold for maximal value of primal iterates.</td>
<td>1e+20</td>
</tr>
<tr>
<td>dual_inf_tol</td>
<td>Desired threshold for the dual infeasibility.</td>
<td>1</td>
</tr>
<tr>
<td>max_cpu_time</td>
<td>Maximum number of CPU seconds.</td>
<td>1e+20</td>
</tr>
<tr>
<td>max_iter</td>
<td>Maximum number of iterations.</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>max_wall_time</td>
<td>Maximum number of walltime clock seconds.</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>mu_target</td>
<td>Desired value of complementarity.</td>
<td>0</td>
</tr>
<tr>
<td>tol</td>
<td>Desired convergence tolerance (relative).</td>
<td>1e-08</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_max</td>
<td>Scaling threshold for the NLP error.</td>
<td>100</td>
</tr>
</tbody>
</table>

### 5.20.5.2 Output

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf_pr_output</td>
<td>Determines what value is printed in the &quot;inf_pr&quot; output column.</td>
<td>original</td>
</tr>
<tr>
<td>print_eval_error</td>
<td>Switch to enable printing information about function evaluation errors into the GAMS listing file.</td>
<td>yes</td>
</tr>
<tr>
<td>print_frequency_iter</td>
<td>Determines at which iteration frequency the summarizing iteration output line should be printed.</td>
<td>1</td>
</tr>
<tr>
<td>print_frequency_time</td>
<td>Determines at which time frequency the summarizing iteration output line should be printed.</td>
<td>0</td>
</tr>
<tr>
<td>print_info_string</td>
<td>Enables printing of additional info string at end of iteration output.</td>
<td>no</td>
</tr>
<tr>
<td>print_level</td>
<td>Output verbosity level.</td>
<td>5</td>
</tr>
<tr>
<td>print_options_mode</td>
<td>format in which to print options documentation</td>
<td>text</td>
</tr>
<tr>
<td>print_timing_statistics</td>
<td>Switch to print timing statistics.</td>
<td>no</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>report_mininfeas_solution</td>
<td>Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.</td>
<td>no</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_advanced_options</td>
<td>whether to print also advanced options</td>
<td>no</td>
</tr>
</tbody>
</table>

5.20.5.3 NLP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_relax_factor</td>
<td>Factor for initial relaxation of the bounds.</td>
<td>1e-10</td>
</tr>
<tr>
<td>check_derivatives_for_naninf</td>
<td>Indicates whether it is desired to check for NaN/Inf in derivative matrices.</td>
<td>no</td>
</tr>
<tr>
<td>fixed_variable_treatment</td>
<td>Determines how fixed variables should be handled.</td>
<td>make_parameter</td>
</tr>
<tr>
<td>honor_original_bounds</td>
<td>Indicates whether final points should be projected into original bounds.</td>
<td>no</td>
</tr>
<tr>
<td>jac_c_constant</td>
<td>Indicates whether to assume that all equality constraints are linear</td>
<td>no</td>
</tr>
<tr>
<td>jac_d_constant</td>
<td>Indicates whether to assume that all inequality constraints are linear</td>
<td>no</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency_detection_with_rhs</td>
<td>Indicates if the right hand sides of the constraints should be considered in addition to gradients during dependency detection</td>
<td>no</td>
</tr>
<tr>
<td>dependency_detector</td>
<td>Indicates which linear solver should be used to detect linearly dependent equality constraints.</td>
<td>none</td>
</tr>
<tr>
<td>kappa_d</td>
<td>Weight for linear damping term (to handle one-sided bounds).</td>
<td>1e-05</td>
</tr>
</tbody>
</table>

5.20.5.4 NLP Scaling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp_scaling_max_gradient</td>
<td>Maximum gradient after NLP scaling.</td>
<td>100</td>
</tr>
<tr>
<td>nlp_scaling_method</td>
<td>Select the technique used for scaling the NLP.</td>
<td>gradient-based if GAMS scaleopt is not set, otherwise none</td>
</tr>
<tr>
<td>nlp_scaling_min_value</td>
<td>Minimum value of gradient-based scaling values.</td>
<td>1e-08</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp_scaling_constr_target_gradient</td>
<td>Target value for constraint function gradient size.</td>
<td>0</td>
</tr>
<tr>
<td>nlp_scaling_obj_target_gradient</td>
<td>Target value for objective function gradient size.</td>
<td>0</td>
</tr>
</tbody>
</table>

5.20.5.5 Initialization
### 5.20.5.6 Warm Start

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm_start_bound_frac</td>
<td>same as bound_frac for the regular initializer</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_bound_push</td>
<td>same as bound_push for the regular initializer</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_init_point</td>
<td>Warm-start for initial point</td>
<td>yes, if run on modified model instance (e.g., from GUSS), otherwise no</td>
</tr>
<tr>
<td>warm_start_mult_bound_push</td>
<td>same as mult_bound_push for the regular initializer</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_mult_init_max</td>
<td>Maximum initial value for the equality multipliers.</td>
<td>1e+06</td>
</tr>
<tr>
<td>warm_start_slack_bound_frac</td>
<td>same as slack_bound_frac for the regular initializer</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_slack_bound_push</td>
<td>same as slack_bound_push for the regular initializer</td>
<td>0.001</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm_start_target_mu</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.20.5.7 Miscellaneous

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>timing_statistics</td>
<td>Indicates whether to measure time spend in components of Ipopt and NLP evaluation</td>
<td>no</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>replace_bounds</td>
<td>Whether all variable bounds should be replaced by inequality constraints</td>
<td>no</td>
</tr>
</tbody>
</table>

### 5.20.5.8 Barrier Parameter Update
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptive_mu_globalization</td>
<td>Globalization strategy for the adaptive mu selection mode.</td>
<td>obj-constr-filter</td>
</tr>
<tr>
<td>barrier_tol_factor</td>
<td>Factor for mu in barrier stop test.</td>
<td>10</td>
</tr>
<tr>
<td>fixed_mu_oracle</td>
<td>Oracle for the barrier parameter when switching to fixed mode.</td>
<td>average_compl</td>
</tr>
<tr>
<td>mu_init</td>
<td>Initial value for the barrier parameter.</td>
<td>0.1</td>
</tr>
<tr>
<td>mu_linear_decrease_factor</td>
<td>Determines linear decrease rate of barrier parameter.</td>
<td>0.2</td>
</tr>
<tr>
<td>mu_max</td>
<td>Maximum value for barrier parameter.</td>
<td>100000</td>
</tr>
<tr>
<td>mu_max_fact</td>
<td>Factor for initialization of maximum value for barrier parameter.</td>
<td>1000</td>
</tr>
<tr>
<td>mu_min</td>
<td>Minimum value for barrier parameter.</td>
<td>1e-11</td>
</tr>
<tr>
<td>mu_oracle</td>
<td>Oracle for a new barrier parameter in the adaptive strategy.</td>
<td>quality-function</td>
</tr>
<tr>
<td>mu_strategy</td>
<td>Update strategy for barrier parameter.</td>
<td>adaptive</td>
</tr>
<tr>
<td>mu_superlinear_decrease_power</td>
<td>Determines superlinear decrease rate of barrier parameter.</td>
<td>1.5</td>
</tr>
<tr>
<td>quality_function_max_section_steps</td>
<td>Maximum number of search steps during direct search procedure determining the optimal centering parameter.</td>
<td>8</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adaptive_mu_kkt_norm_type</td>
<td>Norm used for the KKT error in the adaptive mu globalization strategies.</td>
<td>2-norm-squared</td>
</tr>
<tr>
<td>adaptive_mu_kkt_error_red_fact</td>
<td>Sufficient decrease factor for &quot;kkt-error&quot; globalization strategy.</td>
<td>0.9999</td>
</tr>
<tr>
<td>adaptive_mu_kkt_error_red_iters</td>
<td>Maximum number of iterations requiring sufficient progress.</td>
<td>4</td>
</tr>
<tr>
<td>adaptive_mu_monotone_init_factor</td>
<td>Determines the initial value of the barrier parameter when switching to the monotone mode.</td>
<td>0.8</td>
</tr>
<tr>
<td>adaptive_mu_restore_previous_iterate</td>
<td>Indicates if the previous accepted iterate should be restored if the monotone mode is entered.</td>
<td>no</td>
</tr>
<tr>
<td>filter_margin_fact</td>
<td>Factor determining width of margin for obj-constr-filter adaptive globalization strategy.</td>
<td>1e-05</td>
</tr>
<tr>
<td>filter_max_margin</td>
<td>Maximum width of margin in obj-constr-filter adaptive globalization strategy.</td>
<td>1</td>
</tr>
<tr>
<td>mu_allow_fast_monotone_decrease</td>
<td>Allow skipping of barrier problem if barrier test is already met.</td>
<td>yes</td>
</tr>
<tr>
<td>quality_function_balancing_term</td>
<td>The balancing term included in the quality function for centrality.</td>
<td>none</td>
</tr>
<tr>
<td>quality_function_centrality</td>
<td>The penalty term for centrality that is included in quality function.</td>
<td>none</td>
</tr>
<tr>
<td>quality_function_norm_type</td>
<td>Norm used for components of the quality function.</td>
<td>2-norm-squared</td>
</tr>
<tr>
<td>quality_function_section_qf_tol</td>
<td>Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>quality_function_section_sigma_tol</strong></td>
<td>Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>sigma_max</strong></td>
<td>Maximum value of the centering parameter.</td>
<td>100</td>
</tr>
<tr>
<td><strong>sigma_min</strong></td>
<td>Minimum value of the centering parameter.</td>
<td>1e^-06</td>
</tr>
<tr>
<td><strong>tau_min</strong></td>
<td>Lower bound on fraction-to-the-boundary parameter tau.</td>
<td>0.99</td>
</tr>
</tbody>
</table>

### 5.20.5.9 Line Search

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accept_every_trial_step</strong></td>
<td>Always accept the first trial step.</td>
</tr>
<tr>
<td><strong>alpha_for_y</strong></td>
<td>Method to determine the step size for constraint multipliers (alpha_y).</td>
</tr>
<tr>
<td><strong>alpha_for_y_tol</strong></td>
<td>Tolerance for switching to full equality multiplier steps.</td>
</tr>
<tr>
<td><strong>max_soc</strong></td>
<td>Maximum number of second order correction trial steps at each iteration.</td>
</tr>
<tr>
<td><strong>recalc_y</strong></td>
<td>Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.</td>
</tr>
<tr>
<td><strong>recalc_y_feas_tol</strong></td>
<td>Feasibility threshold for recomputation of multipliers.</td>
</tr>
<tr>
<td><strong>soc_method</strong></td>
<td>Ways to apply second order correction</td>
</tr>
<tr>
<td><strong>watchdog_shortened_iter_trigger</strong></td>
<td>Number of shortened iterations that trigger the watchdog.</td>
</tr>
<tr>
<td><strong>watchdog_trial_iter_max</strong></td>
<td>Maximum number of watchdog iterations.</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accept_after_max_steps</strong></td>
<td>Accept a trial point after maximal this number of steps even if it does not satisfy line search conditions.</td>
</tr>
<tr>
<td><strong>alpha_min_frac</strong></td>
<td>Safety factor for the minimal step size (before switching to restoration phase).</td>
</tr>
<tr>
<td><strong>alpha_red_factor</strong></td>
<td>Fractional reduction of the trial step size in the backtracking line search.</td>
</tr>
<tr>
<td><strong>constraintViolationNormType</strong></td>
<td>Norm to be used for the constraint violation in the line search.</td>
</tr>
<tr>
<td><strong>corrector_complAvgRedFact</strong></td>
<td>Complementarity tolerance factor for accepting corrector step.</td>
</tr>
<tr>
<td><strong>corrector_type</strong></td>
<td>The type of corrector steps that should be taken.</td>
</tr>
<tr>
<td><strong>delta</strong></td>
<td>Multiplier for constraint violation in the switching rule.</td>
</tr>
<tr>
<td><strong>eta_phi</strong></td>
<td>Relaxation factor in the Armijo condition.</td>
</tr>
<tr>
<td><strong>filter_reset_trigger</strong></td>
<td>Number of iterations that trigger the filter reset.</td>
</tr>
<tr>
<td><strong>gamma_phi</strong></td>
<td>Relaxation factor in the filter margin for the barrier function.</td>
</tr>
<tr>
<td><strong>gamma_theta</strong></td>
<td>Relaxation factor in the filter margin for the constraint violation.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>kappa_sigma</td>
<td>Factor limiting the deviation of dual variables from primal estimates.</td>
</tr>
<tr>
<td>kappa_soc</td>
<td>Factor in the sufficient reduction rule for second order correction.</td>
</tr>
<tr>
<td>line_search_method</td>
<td>Globalization method used in backtracking line search</td>
</tr>
<tr>
<td>max_filter_resets</td>
<td>Maximal allowed number of filter resets</td>
</tr>
<tr>
<td>nu_inc</td>
<td>Increment of the penalty parameter.</td>
</tr>
<tr>
<td>nu_init</td>
<td>Initial value of the penalty parameter.</td>
</tr>
<tr>
<td>obj_max_inc</td>
<td>Determines the upper bound on the acceptable increase of barrier objective function.</td>
</tr>
<tr>
<td>rho</td>
<td>Value in penalty parameter update formula.</td>
</tr>
<tr>
<td>s_phi</td>
<td>Exponent for linear barrier function model in the switching rule.</td>
</tr>
<tr>
<td>s_theta</td>
<td>Exponent for current constraint violation in the switching rule.</td>
</tr>
<tr>
<td>skip_corr_if_neg_curv</td>
<td>Whether to skip the corrector step in negative curvature iteration.</td>
</tr>
<tr>
<td>skip_corr_in_monotone_mode</td>
<td>Whether to skip the corrector step during monotone barrier parameter mode.</td>
</tr>
<tr>
<td>slack_move</td>
<td>Correction size for very small slacks.</td>
</tr>
<tr>
<td>theta_max_fact</td>
<td>Determines upper bound for constraint violation in the filter.</td>
</tr>
<tr>
<td>theta_min_fact</td>
<td>Determines constraint violation threshold in the switching rule.</td>
</tr>
<tr>
<td>tiny_step_tol</td>
<td>Tolerance for detecting numerically insignificant steps.</td>
</tr>
<tr>
<td>tiny_step_y_tol</td>
<td>Tolerance for quitting because of numerically insignificant steps.</td>
</tr>
</tbody>
</table>

### 5.20.5.10 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hslolib</td>
<td>Name of library containing HSL routines for load at runtime</td>
<td>libhsl.so (Linux), libhsl.dylib (macOS), libhsl.dll (Windows)</td>
</tr>
<tr>
<td>linear_scaling_on_demand</td>
<td>Flag indicating that linear scaling is only done if it seems required.</td>
<td>yes</td>
</tr>
<tr>
<td>linear_solver</td>
<td>Linear solver used for step computations.</td>
<td>ma27, if IpoptH, otherwise mumps</td>
</tr>
<tr>
<td>linear_system_scaling</td>
<td>Method for scaling the linear system.</td>
<td>mc19, if IpoptH, otherwise none</td>
</tr>
<tr>
<td>pardisolib</td>
<td>Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime</td>
<td>libpardiso.so (Linux), libpardiso.dylib (macOS), libpardiso.dll (Windows)</td>
</tr>
</tbody>
</table>

### 5.20.5.11 Step Calculation
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast_step_computation</td>
<td>Indicates if the linear system should be solved quickly.</td>
<td>no</td>
</tr>
<tr>
<td>first_hessian_perturbation</td>
<td>Size of first x-s perturbation tried.</td>
<td>0.0001</td>
</tr>
<tr>
<td>jacobian_regularization_value</td>
<td>Size of the regularization for rank-deficient constraint Jacobians.</td>
<td>1e-08</td>
</tr>
<tr>
<td>max_hessian_perturbation</td>
<td>Maximum value of regularization parameter for handling negative curvature.</td>
<td>1e+20</td>
</tr>
<tr>
<td>max_refinement_steps</td>
<td>Maximum number of iterative refinement steps per linear system solve.</td>
<td>10</td>
</tr>
<tr>
<td>mehrotra_algorithm</td>
<td>Indicates whether to do Mehrotra's predictor-corrector algorithm.</td>
<td>no</td>
</tr>
<tr>
<td>min_hessian_perturbation</td>
<td>Smallest perturbation of the Hessian block.</td>
<td>1e-20</td>
</tr>
<tr>
<td>min_refinement_steps</td>
<td>Minimum number of iterative refinement steps per linear system solve.</td>
<td>1</td>
</tr>
<tr>
<td>neg_curv_test_reg</td>
<td>Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014).</td>
<td>yes</td>
</tr>
<tr>
<td>neg_curv_test_tol</td>
<td>Tolerance for heuristic to ignore wrong inertia.</td>
<td>0</td>
</tr>
<tr>
<td>perturb_dec_fact</td>
<td>Decrease factor for x-s perturbation.</td>
<td>0.333333</td>
</tr>
<tr>
<td>perturb_inc_fact</td>
<td>Increase factor for x-s perturbation.</td>
<td>8</td>
</tr>
<tr>
<td>perturb_inc_fact_first</td>
<td>Increase factor for x-s perturbation for very first perturbation.</td>
<td>100</td>
</tr>
<tr>
<td>jacobian_regularization_exponent</td>
<td>Exponent for mu in the regularization for rank-deficient constraint Jacobians.</td>
<td>0.25</td>
</tr>
<tr>
<td>perturb_always_cd</td>
<td>Active permanent perturbation of constraint linearization.</td>
<td>no</td>
</tr>
<tr>
<td>residual_improvement_factor</td>
<td>Minimal required reduction of residual test ratio in iterative refinement.</td>
<td>1</td>
</tr>
<tr>
<td>residual_ratio_max</td>
<td>Iterative refinement tolerance.</td>
<td>1e-10</td>
</tr>
<tr>
<td>residual_ratio_singular</td>
<td>Threshold for declaring linear system singular after failed iterative refinement.</td>
<td>1e-05</td>
</tr>
</tbody>
</table>

### 5.20.5.12 Restoration Phase

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_mult_reset_threshold</td>
<td>Threshold for resetting bound multipliers after the restoration phase.</td>
<td>1000</td>
</tr>
<tr>
<td>constr_mult_reset_threshold</td>
<td>Threshold for resetting equality and inequality multipliers after restoration phase.</td>
<td>0</td>
</tr>
<tr>
<td>evaluate_orig_obj_at_resto_trial</td>
<td>Determines if the original objective function should be evaluated at restoration phase trial points.</td>
<td>yes</td>
</tr>
<tr>
<td>expect_infeasible_problem</td>
<td>Enable heuristics to quickly detect an infeasible problem.</td>
<td>no</td>
</tr>
<tr>
<td>expect_infeasible_problem_ctol</td>
<td>Threshold for disabling &quot;expect_infeasible_problem&quot; option.</td>
<td>0.001</td>
</tr>
<tr>
<td>expect_infeasible_problem_ytol</td>
<td>Multiplier threshold for activating &quot;expect_infeasible_problem&quot; option.</td>
<td>1e+08</td>
</tr>
<tr>
<td>required_infeasibility_reduction</td>
<td>Required reduction of infeasibility before leaving restoration phase.</td>
<td>0.9</td>
</tr>
</tbody>
</table>
### 5.20.5.13 Hessian Approximation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian_approximation</td>
<td>Indicates what Hessian information is to be used.</td>
<td>exact</td>
</tr>
<tr>
<td>limited_memory_init_val</td>
<td>Value for B0 in low-rank update.</td>
<td>1</td>
</tr>
<tr>
<td>limited_memory_init_val_max</td>
<td>Upper bound on value for B0 in low-rank update.</td>
<td>1e+08</td>
</tr>
<tr>
<td>limited_memory_init_val_min</td>
<td>Lower bound on value for B0 in low-rank update.</td>
<td>1e-08</td>
</tr>
<tr>
<td>limited_memory_initialization</td>
<td>Initialization strategy for the limited memory quasi-Newton approximation.</td>
<td>scalar1</td>
</tr>
<tr>
<td>limited_memory_max_history</td>
<td>Maximum size of the history for the limited quasi-Newton Hessian approximation.</td>
<td>6</td>
</tr>
<tr>
<td>limited_memory_max_skipping</td>
<td>Threshold for successive iterations where update is skipped.</td>
<td>2</td>
</tr>
<tr>
<td>limited_memory_special_for_resto</td>
<td>Determines if the quasi-Newton updates should be special during the restoration phase.</td>
<td>no</td>
</tr>
<tr>
<td>limited_memory_update_type</td>
<td>Quasi-Newton update formula for the limited memory quasi-Newton approximation.</td>
<td>bfgs</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian_approximation_space</td>
<td>Indicates in which subspace the Hessian information is to be approximated.</td>
<td>nonlinear-variables</td>
</tr>
<tr>
<td>limited_memory_aug_solver</td>
<td>Strategy for solving the augmented system for low-rank Hessian.</td>
<td>sherman-morrison</td>
</tr>
</tbody>
</table>

### 5.20.5.14 MA27 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma27_la_init_factor</td>
<td>Real workspace memory for MA27.</td>
<td>5</td>
</tr>
<tr>
<td>ma27_liw_init_factor</td>
<td>Integer workspace memory for MA27.</td>
<td>5</td>
</tr>
</tbody>
</table>
5.20 IPOPT and IPOPTH

### 5.20.5.15 MA57 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma57_automatic_scaling</td>
<td>Controls whether to enable automatic scaling in MA57</td>
<td>no</td>
</tr>
<tr>
<td>ma57_block_size</td>
<td>Controls block size used by Level 3 BLAS in MA57BD</td>
<td>16</td>
</tr>
<tr>
<td>ma57_node_amalgamation</td>
<td>Node amalgamation parameter</td>
<td>16</td>
</tr>
<tr>
<td>ma57_pivot_order</td>
<td>Controls pivot order in MA57</td>
<td>5</td>
</tr>
<tr>
<td>ma57_pivot</td>
<td>Pivot tolerance for the linear solver MA57</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma57_pivotolmax</td>
<td>Maximum pivot tolerance for the linear solver MA57</td>
<td>0.0001</td>
</tr>
<tr>
<td>ma57_pre_alloc</td>
<td>Safety factor for work space memory allocation for the linear solver MA57</td>
<td>1.05</td>
</tr>
<tr>
<td>ma57_print_level</td>
<td>Debug printing level for the linear solver MA57</td>
<td>0</td>
</tr>
<tr>
<td>ma57_small_pivot_flag</td>
<td>Handling of small pivots</td>
<td>0</td>
</tr>
</tbody>
</table>

5.20.5.16 MA77 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma77_buffer_lpage</td>
<td>Number of scalars per MA77 in-core buffer page in the out-of-core solver MA77</td>
<td>4096</td>
</tr>
<tr>
<td>ma77_buffer_npage</td>
<td>Number of pages that make up MA77 buffer</td>
<td>1600</td>
</tr>
<tr>
<td>ma77_file_size</td>
<td>Target size of each temporary file for MA77, scalars per type</td>
<td>2097152</td>
</tr>
<tr>
<td>ma77_maxstore</td>
<td>Maximum storage size for MA77 in-core mode</td>
<td>0</td>
</tr>
<tr>
<td>ma77_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma77_order</td>
<td>Controls type of ordering used by MA77</td>
<td>metis</td>
</tr>
<tr>
<td>ma77_print_level</td>
<td>Debug printing level for the linear solver MA77</td>
<td>-1</td>
</tr>
<tr>
<td>ma77_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma77_static</td>
<td>Static Pivoting Threshold</td>
<td>0</td>
</tr>
<tr>
<td>ma77_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>ma77_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

### 5.20.5.17 MA86 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma86_nemin</td>
<td>Node Amalgamation parameter</td>
<td>32</td>
</tr>
<tr>
<td>ma86_order</td>
<td>Controls type of ordering</td>
<td>auto</td>
</tr>
<tr>
<td>ma86_print_level</td>
<td>Debug printing level</td>
<td>-1</td>
</tr>
<tr>
<td>ma86_scaling</td>
<td>Controls scaling of matrix</td>
<td>mc64</td>
</tr>
<tr>
<td>ma86_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma86_static</td>
<td>Static Pivoting Threshold</td>
<td>0</td>
</tr>
<tr>
<td>ma86_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma86_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

### 5.20.5.18 MA97 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma97_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma97_order</td>
<td>Controls type of ordering</td>
<td>auto</td>
</tr>
<tr>
<td>ma97_print_level</td>
<td>Debug printing level</td>
<td>-1</td>
</tr>
<tr>
<td>ma97_scaling</td>
<td>Specifies strategy for scaling</td>
<td>dynamic</td>
</tr>
<tr>
<td>ma97_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma97_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma97_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma97_scaling1</td>
<td>First scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling2</td>
<td>Second scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling3</td>
<td>Third scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_solve_blas3</td>
<td>Controls if blas2 or blas3 routines are used for solve</td>
<td>no</td>
</tr>
<tr>
<td>ma97_switch1</td>
<td>First switch, determine when ma97_scaling1 is enabled.</td>
<td>od_hd_reuse</td>
</tr>
<tr>
<td>ma97_switch2</td>
<td>Second switch, determine when ma97_scaling2 is enabled.</td>
<td>never</td>
</tr>
<tr>
<td>ma97_switch3</td>
<td>Third switch, determine when ma97_scaling3 is enabled.</td>
<td>never</td>
</tr>
</tbody>
</table>

### 5.20.5.19 Pardiso (pardiso-project.org) Linear Solver
### 5.20 IPOPT and IPOPTH

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pardiso_matching_strategy</strong></td>
<td>Matching strategy to be used by Pardiso</td>
<td>complete+2x2</td>
</tr>
<tr>
<td><strong>pardiso_max_iterative_refinement_steps</strong></td>
<td>Limit on number of iterative refinement steps.</td>
<td>0</td>
</tr>
<tr>
<td><strong>pardiso_msglvl</strong></td>
<td>Pardiso message level</td>
<td>0</td>
</tr>
<tr>
<td><strong>pardiso_order</strong></td>
<td>Controls the fill-in reduction ordering algorithm for the input matrix.</td>
<td>metis</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pardiso_iter_coarse_size</strong></td>
<td>Maximum Size of Coarse Grid Matrix</td>
<td>5000</td>
</tr>
<tr>
<td><strong>pardiso_iter_dropping_factor</strong></td>
<td>dropping value for incomplete factor</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>pardiso_iter_dropping_schur</strong></td>
<td>dropping value for sparsify schur complement factor</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>pardiso_iter_inverse_norm_factor</strong></td>
<td>Maximum Size of Grid Levels</td>
<td>5e+06</td>
</tr>
<tr>
<td><strong>pardiso_iter_max_levels</strong></td>
<td>max fill for each row</td>
<td>10</td>
</tr>
<tr>
<td><strong>pardiso_iter_max_row_fill</strong></td>
<td>Relative Residual Convergence</td>
<td>1e-06</td>
</tr>
<tr>
<td><strong>pardiso_iterative</strong></td>
<td>Switch for iterative solver in Pardiso library</td>
<td>no</td>
</tr>
<tr>
<td><strong>pardiso_max_droptol_corrections</strong></td>
<td>Maximal number of decreases of drop tolerance during one solve.</td>
<td>4</td>
</tr>
<tr>
<td><strong>pardiso_max_iter</strong></td>
<td>Maximum number of Krylov-Subspace Iteration</td>
<td>500</td>
</tr>
<tr>
<td><strong>pardiso_redo_symbolic_fact_only_if_inertia_wrong</strong></td>
<td>Toggle for handling case when elements were perturbed by Pardiso.</td>
<td>no</td>
</tr>
<tr>
<td><strong>pardiso_repeated_perturbation_means_singular</strong></td>
<td>Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization.</td>
<td>no</td>
</tr>
<tr>
<td><strong>pardiso_skip_inertia_check</strong></td>
<td>Whether to pretend that inertia is correct.</td>
<td>no</td>
</tr>
</tbody>
</table>

---

### 5.20.5.20 Pardiso (MKL) Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pardisomkl_matching_strategy</strong></td>
<td>Matching strategy to be used by Pardiso</td>
<td>complete+2x2</td>
</tr>
<tr>
<td><strong>pardisomkl_max_iterative_refinement_steps</strong></td>
<td>Limit on number of iterative refinement steps.</td>
<td>1</td>
</tr>
<tr>
<td><strong>pardisomkl_msglvl</strong></td>
<td>Pardiso message level</td>
<td>0</td>
</tr>
<tr>
<td><strong>pardisomkl_order</strong></td>
<td>Controls the fill-in reduction ordering algorithm for the input matrix.</td>
<td>metis</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pardisomkl_redo_symbolic_fact_only_if_inertia_wrong</strong></td>
<td>Toggle for handling case when elements were perturbed by Pardiso.</td>
<td>no</td>
</tr>
<tr>
<td><strong>pardisomkl_repeated_perturbation_means_singular</strong></td>
<td>Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization.</td>
<td>no</td>
</tr>
<tr>
<td><strong>pardisomkl_skip_inertia_check</strong></td>
<td>Whether to pretend that inertia is correct.</td>
<td>no</td>
</tr>
</tbody>
</table>
### 5.20.5.21 Mumps Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mumps_mem_percent</td>
<td>Percentage increase in the estimated working space for MUMPS.</td>
<td>1000</td>
</tr>
<tr>
<td>mumps_permuting_scaling</td>
<td>Controls permuting and scaling in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivot_order</td>
<td>Controls pivot order in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivotol</td>
<td>Pivot tolerance for the linear solver MUMPS.</td>
<td>1e-06</td>
</tr>
<tr>
<td>mumps_pivotolmax</td>
<td>Maximum pivot tolerance for the linear solver MUMPS.</td>
<td>0.1</td>
</tr>
<tr>
<td>mumps_print_level</td>
<td>Debug printing level for the linear solver MUMPS.</td>
<td>0</td>
</tr>
<tr>
<td>mumps_scaling</td>
<td>Controls scaling in MUMPS</td>
<td>77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options for expert users</th>
</tr>
</thead>
<tbody>
<tr>
<td>mumps_dep_tol</td>
</tr>
</tbody>
</table>

### 5.20.6 Detailed Options Description

**accept_after_max_steps** *(advanced)*: Accept a trial point after maximal this number of steps even if it does not satisfy line search conditions. ←

Setting this to -1 disables this option.

Range: {-1, ..., ∞}

Default: -1

**accept_every_trial_step**: Always accept the first trial step. ←

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

Range: yes, no

Default: no

**acceptable_compl_inf_tol**: "Acceptance" threshold for the complementarity conditions. ←

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.

Range: (0, ∞]

Default: 0.01

**acceptable_constr_viol_tol**: "Acceptance" threshold for the constraint violation. ←

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.

Range: (0, ∞]

Default: 0.01
acceptable_dual_inf_tol: "Acceptance" threshold for the dual infeasibility. 

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.

Range: \((0, \infty]\)

Default: \(1e+10\)

acceptable_iter: Number of "acceptable" iterates before triggering termination. 

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

Range: \(\{0, ..., \infty\}\)

Default: 0

acceptable_obj_change_tol: "Acceptance" stopping criterion based on objective function change. 

If the relative change of the objective function (scaled by Max\(1,|f(x)|\)) is less than this value, this part of the acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

Range: \([0, \infty]\)

Default: \(1e+20\)

acceptable_tol: "Acceptable" convergence tolerance (relative). 

Determines which (scaled) overall optimality error is considered to be "acceptable". There are two levels of termination criteria. If the usual "desired" tolerances (see tol, dual_inf_tol etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

Range: \((0, \infty]\)

Default: \(1e-06\)

adaptive_mu_globalization: Globalization strategy for the adaptive mu selection mode. 

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>kkt-error</td>
<td>nonmonotone decrease of kkt-error</td>
</tr>
<tr>
<td>obj-constr-filter</td>
<td>2-dim filter for objective and constraint violation</td>
</tr>
<tr>
<td>never-monotone-mode</td>
<td>disables globalization</td>
</tr>
</tbody>
</table>

Default: obj-constr-filter
adaptive_mu_kkt_norm_type (advanced): Norm used for the KKT error in the adaptive mu globalization strategies.

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this option is also used in the QualityFunctionMuOracle.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-norm</td>
<td>use the 1-norm (abs sum)</td>
</tr>
<tr>
<td>2-norm-squared</td>
<td>use the 2-norm squared (sum of squares)</td>
</tr>
<tr>
<td>max-norm</td>
<td>use the infinity norm (max)</td>
</tr>
<tr>
<td>2-norm</td>
<td>use 2-norm</td>
</tr>
</tbody>
</table>

Default: 2-norm-squared

adaptive_mu_kkterror_red_fact (advanced): Sufficient decrease factor for "kkt-error" globalization strategy.

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: (0, 1)

Default: 0.9999

adaptive_mu_kkterror_red_iters (advanced): Maximum number of iterations requiring sufficient progress.

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

Range: {0, ..., ∞}

Default: 4

adaptive_mu_monotone_init_factor (advanced): Determines the initial value of the barrier parameter when switching to the monotone mode.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

Range: (0, ∞]

Default: 0.8

adaptive_mu_restore_previous_iterate (advanced): Indicates if the previous accepted iterate should be restored if the monotone mode is entered.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

Range: yes, no

Default: no

alpha_for_y: Method to determine the step size for constraint multipliers (alpha_y).

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>primal</td>
<td>use primal step size</td>
</tr>
<tr>
<td>bound-mult</td>
<td>use step size for the bound multipliers (good for LPs)</td>
</tr>
<tr>
<td>min</td>
<td>use the min of primal and bound multipliers</td>
</tr>
<tr>
<td>max</td>
<td>use the max of primal and bound multipliers</td>
</tr>
<tr>
<td>full</td>
<td>take a full step of size one</td>
</tr>
<tr>
<td>min-dual-infeas</td>
<td>choose step size minimizing new dual infeasibility</td>
</tr>
<tr>
<td>safer-min-dual-infeas</td>
<td>like &quot;min-dual-infeas&quot;, but safeguarded by &quot;min&quot; and &quot;max&quot;</td>
</tr>
<tr>
<td>primal-and-full</td>
<td>use the primal step size, and full step if $\Delta x \leq \alpha_{\text{for}_y_tol}$</td>
</tr>
<tr>
<td>dual-and-full</td>
<td>use the dual step size, and full step if $\Delta x \leq \alpha_{\text{for}_y_tol}$</td>
</tr>
<tr>
<td>acceptor</td>
<td>Call LSAcceptor to get step size for $y$</td>
</tr>
</tbody>
</table>

Default: primal

**alpha_for_y_tol:** Tolerance for switching to full equality multiplier steps.  
This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

Range: $[0, \infty]$  
Default: 10

**alpha_min_frac** *(advanced):* Safety factor for the minimal step size (before switching to restoration phase).  
This is gamma_alpha in Eqn. (20) in the implementation paper.

Range: $(0, 1)$  
Default: 0.05

**alpha_red_factor** *(advanced):* Fractional reduction of the trial step size in the backtracking line search.

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: $(0, 1)$  
Default: 0.5

**barrier_tol_factor:** Factor for mu in barrier stop test.

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. This is kappa_epsilon in implementation paper.

Range: $(0, \infty]$  
Default: 10

**bound_frac:** Desired minimum relative distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: $(0, 0.5]$  
Default: 0.01

**bound_mult_init_method:** Initialization method for bound multipliers

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, the each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>set all bound multipliers to the value of bound_mult_init_val</td>
</tr>
<tr>
<td>mu-based</td>
<td>initialize to mu_init/x_slack</td>
</tr>
</tbody>
</table>

Default: constant

**bound_mult_init_val**: Initial value for the bound multipliers. ↔

All dual variables corresponding to bound constraints are initialized to this value.

Range: \((0, \infty]\)

Default: 1

**bound_mult_reset_threshold**: Threshold for resetting bound multipliers after the restoration phase. ↔

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

Range: \([0, \infty]\)

Default: 1000

**bound_push**: Desired minimum absolute distance from the initial point to bound. ↔

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Range: \((0, \infty]\)

Default: 0.01

**bound_relax_factor**: Factor for initial relaxation of the bounds. ↔

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. Additional, the constraint violation tolerance constr_viol_tol is used to bound the relaxation by an absolute value. If it is set to zero, then then bounds relaxation is disabled. See Eqn.(35) in implementation paper. Note that the constraint violation reported by Ipopt at the end of the solution process does not include violations of the original (non-relaxed) variable bounds. See also option honor_original_bounds.

Range: \([0, \infty]\)

Default: 1e-10

**check_derivatives_for_naninf**: Indicates whether it is desired to check for Nan/Inf in derivative matrices ↔
Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE DETAILED; so beware of large output!

Range: yes, no
Default: no

**compl_inf_tol:** Desired threshold for the complementarity conditions. ↔

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

Range: $(0, \infty]$  
Default: $0.0001$

**constr_mult_init_max:** Maximum allowed least-square guess of constraint multipliers. ↔

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

Range: $[0, \infty]$  
Default: $1000$

**constr_mult_reset_threshold:** Threshold for resetting equality and inequality multipliers after restoration phase. ↔

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

Range: $[0, \infty]$  
Default: $0$

**constr_viol_tol:** Desired threshold for the constraint and variable bound violation. ↔

Absolute tolerance on the constraint and variable bound violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold. If option bound_relax_factor is not zero 0, then Ipopt relaxes given variable bounds. The value of constr_viol_tol is used to restrict the absolute amount of this bound relaxation.

Range: $(0, \infty]$  
Default: $1e-06$

**constraintViolationNormType (advanced):** Norm to be used for the constraint violation in the line search. ↔

Determines which norm should be used when the algorithm computes the constraint violation in the line search.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-norm</td>
<td>use the 1-norm</td>
</tr>
<tr>
<td>2-norm</td>
<td>use the 2-norm</td>
</tr>
<tr>
<td>max-norm</td>
<td>use the infinity norm</td>
</tr>
</tbody>
</table>

Default: 1-norm

**corrector_compl_avrg_red_fact (advanced):** Complementarity tolerance factor for accepting corrector step. ←

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. Changing this option is experimental.

Range: \((0, \infty]\)

Default: 1

**corrector_type (advanced):** The type of corrector steps that should be taken. ←

If “mu_strategy” is “adaptive”, this option determines what kind of corrector steps should be tried. Changing this option is experimental.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no corrector</td>
</tr>
<tr>
<td>affine</td>
<td>corrector step towards (\mu=0)</td>
</tr>
<tr>
<td>primal-dual</td>
<td>corrector step towards current (\mu)</td>
</tr>
</tbody>
</table>

Default: none

**delta (advanced):** Multiplier for constraint violation in the switching rule. ←

See Eqn. (19) in the implementation paper.

Range: \((0, \infty]\)

Default: 1

**dependency_detection_with_rhs (advanced):** Indicates if the right hand sides of the constraints should be considered in addition to gradients during dependency detection ←

Range: yes, no

Default: no

**dependency_detector (advanced):** Indicates which linear solver should be used to detect linearly dependent equality constraints. ←

This is experimental and does not work well.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>don't check; no extra work at beginning</td>
</tr>
<tr>
<td>mumps</td>
<td>use MUMPS</td>
</tr>
</tbody>
</table>
diverging_iterates_tol: Threshold for maximal value of primal iterates. ↔

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

Range: \((0, \infty]\)

Default: \(1e+20\)

dual_inf_tol: Desired threshold for the dual infeasibility. ↔

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

Range: \((0, \infty]\)

Default: 1

eta_phi (advanced): Relaxation factor in the Armijo condition. ↔

See Eqn. (20) in the implementation paper.

Range: \((0, 0.5]\)

Default: \(1e-08\)

evaluate_orig_obj_at_resto_trial: Determines if the original objective function should be evaluated at restoration phase trial points. ↔

Enabling this option makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

Range: yes, no

Default: yes

expect_infeasible_problem: Enable heuristics to quickly detect an infeasible problem. ↔

This option is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

Range: yes, no

Default: no

expect_infeasible_problem_ctol: Threshold for disabling "expect_infeasible_problem" option. ↔
If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this option is set to 0.

Range: \([0, \infty]\)
Default: 0.001

**expect_infeasible_problem_ytol:** Multiplier threshold for activating "expect_infeasible_problem" option.

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

Range: \((0, \infty]\)
Default: 1e+08

**fast_step_computation:** Indicates if the linear system should be solved quickly.

If enabled, the algorithm assumes that the linear system that is solved to obtain the search direction is solved sufficiently well. In that case, no residuals are computed to verify the solution and the computation of the search direction is a little faster.

Range: yes, no
Default: no

**filter_margin_fact (advanced):** Factor determining width of margin for obj-constr-filter adaptive globalization strategy.

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: \(\text{new obj} < \text{(filter obj)} - \text{filter_margin_fact} \times \text{(new constr-viol)}\) OR \(\text{new constr-viol} < \text{(filter constr-viol)} - \text{filter_margin_fact} \times \text{(new constr-viol)}\). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: \((0, 1]\)
Default: 1e-05

**filter_max_margin (advanced):** Maximum width of margin in obj-constr-filter adaptive globalization strategy.

Range: \((0, \infty]\)
Default: 1

**filter_reset_trigger (advanced):** Number of iterations that trigger the filter reset.

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

Range: \(\{1, ..., \infty\}\)
Default: 5

**first_hessian_perturbation:** Size of first x-s perturbation tried.

The first value tried for the x-s perturbation in the inertia correction scheme. This is \(\delta_0\) in the implementation paper.

Range: \((0, \infty]\)
Default: 0.0001

**fixed_mu_oracle:** Oracle for the barrier parameter when switching to fixed mode.

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy").
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>probing</td>
<td>Mehrotra’s probing heuristic</td>
</tr>
<tr>
<td>loqo</td>
<td>LOQO’s centrality rule</td>
</tr>
<tr>
<td>quality-function</td>
<td>minimize a quality function</td>
</tr>
<tr>
<td>average_compl</td>
<td>base on current average complementarity</td>
</tr>
</tbody>
</table>

Default: average_compl

**fixed_variable_treatment**: Determines how fixed variables should be handled. ←

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter(nodual)" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to" bound_relax_factor"). For all but "make_parameter_nodual", bound multipliers are computed for the fixed variables.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>make_parameter</td>
<td>Remove fixed variable from optimization variables</td>
</tr>
<tr>
<td>make_parameter_nodual</td>
<td>Remove fixed variable from optimization variables and do not compute bound multipliers for fixed variables</td>
</tr>
<tr>
<td>make_constraint</td>
<td>Add equality constraints fixing variables</td>
</tr>
<tr>
<td>relax_bounds</td>
<td>Relax fixing bound constraints</td>
</tr>
</tbody>
</table>

Default: make_parameter

**gamma_phi (advanced)**: Relaxation factor in the filter margin for the barrier function. ←

See Eqn. (18a) in the implementation paper.

Range: (0, 1)

Default: 1e-08

**gamma_theta (advanced)**: Relaxation factor in the filter margin for the constraint violation. ←

See Eqn. (18b) in the implementation paper.

Range: (0, 1)

Default: 1e-05

**hessian_approximation**: Indicates what Hessian information is to be used. ←

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>exact</td>
<td>Use second derivatives provided by the NLP.</td>
</tr>
<tr>
<td>limited-memory</td>
<td>Perform a limited-memory quasi-Newton approximation</td>
</tr>
</tbody>
</table>

Default: exact
**hessian_approximation_space** *(advanced):* Indicates in which subspace the Hessian information is to be approximated.  

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonlinear-variables</td>
<td>only in space of nonlinear variables.</td>
</tr>
<tr>
<td>all-variables</td>
<td>in space of all variables (without slacks)</td>
</tr>
</tbody>
</table>

Default: nonlinear-variables

**honor_original_bounds:** Indicates whether final points should be projected into original bounds.  

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization. Note that violations of constraints and complementarity reported by Ipopt at the end of the solution process are for the non-projected point.

Range: yes, no

Default: no

**hsllib:** Name of library containing HSL routines for load at runtime

Range: string

Default: libhsl.so (Linux), libhsl.dylib (macOS), libhsl.dll (Windows)

**inf_pr_output:** Determines what value is printed in the "inf_pr" output column.

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal</td>
<td>max-norm of violation of internal equality constraints</td>
</tr>
<tr>
<td>original</td>
<td>maximal constraint violation in original NLP</td>
</tr>
</tbody>
</table>

Default: original

**jac_e_constant:** Indicates whether to assume that all equality constraints are linear

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

Range: yes, no

Default: no

**jac_d_constant:** Indicates whether to assume that all inequality constraints are linear

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

Range: yes, no

Default: no
**jacobian_regularization_exponent** *(advanced)*: Exponent for $\mu$ in the regularization for rank-deficient constraint Jacobians. 

This is $\kappa_c$ in the implementation paper.

Range: $[0, \infty]$  
Default: 0.25

**jacobian_regularization_value**: Size of the regularization for rank-deficient constraint Jacobians. 

This is $\bar{\delta_c}$ in the implementation paper.

Range: $[0, \infty]$  
Default: $1 \times 10^{-8}$

**kappa_d** *(advanced)*: Weight for linear damping term (to handle one-sided bounds). 

See Section 3.7 in implementation paper.

Range: $[0, \infty]$  
Default: $1 \times 10^{-5}$

**kappa_sigma** *(advanced)*: Factor limiting the deviation of dual variables from primal estimates. 

If the dual variables deviate from their primal estimates, a correction is performed. See Eqn. (16) in the implementation paper. Setting the value to less than 1 disables the correction.

Range: $(0, \infty]$  
Default: $1 \times 10^{10}$

**kappa_soc** *(advanced)*: Factor in the sufficient reduction rule for second order correction. 

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. See Step A-5.9 of Algorithm A in the implementation paper.

Range: $(0, \infty]$  
Default: 0.99

**least_square_init_duals**: Least square initialization of all dual variables

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least $\text{bound\_mult\_init\_val}$. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound\_mult\_init\_method".

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>use $\text{bound_mult_init_val}$ and least-square equality constraint multipliers</td>
</tr>
<tr>
<td>yes</td>
<td>overwrite user-provided point with least-square estimates</td>
</tr>
</tbody>
</table>

Default: no
least_square_init_primal: Least square initialization of the primal variables

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s) to fit the linearized equality and inequality constraints. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>take user-provided point</td>
</tr>
<tr>
<td>yes</td>
<td>overwrite user-provided point with least-square estimates</td>
</tr>
</tbody>
</table>

Default: no

limited_memory_augSolver (advanced): Strategy for solving the augmented system for low-rank Hessian.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sherman-morrison</td>
<td>use Sherman-Morrison formula</td>
</tr>
<tr>
<td>extended</td>
<td>use an extended augmented system</td>
</tr>
</tbody>
</table>

Default: sherman-morrison

limited_memory_init_val: Value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1

limited_memory_init_val_max: Upper bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1e+08

limited_memory_init_val_min: Lower bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1e-08

limited_memory_initialization: Initialization strategy for the limited memory quasi-Newton approximation.

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.
### 5.20 IPOPT and IPOPTH

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar1</td>
<td>( \sigma = s^\top T_y/s^\top T_s )</td>
</tr>
<tr>
<td>scalar2</td>
<td>( \sigma = y^\top T_y/s^\top T_y )</td>
</tr>
<tr>
<td>scalar3</td>
<td>arithmetic average of scalar1 and scalar2</td>
</tr>
<tr>
<td>scalar4</td>
<td>geometric average of scalar1 and scalar2</td>
</tr>
<tr>
<td>constant</td>
<td>( \sigma = \text{limited_memory_init_val} )</td>
</tr>
</tbody>
</table>

Default: scalar1

**limited_memory_max_history**: Maximum size of the history for the limited quasi-Newton Hessian approximation. ←

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

Range: \( \{0, ..., \infty\} \)

Default: 6

**limited_memory_max_skipping**: Threshold for successive iterations where update is skipped. ←

If the update is skipped more than this number of successive iterations, the quasi-Newton approximation is reset.

Range: \( \{1, ..., \infty\} \)

Default: 2

**limited_memory_special_for_resto**: Determines if the quasi-Newton updates should be special during the restoration phase. ←

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

Range: yes, no

Default: no

**limited_memory_update_type**: Quasi-Newton update formula for the limited memory quasi-Newton approximation. ←

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfgs</td>
<td>BFGS update (with skipping)</td>
</tr>
<tr>
<td>sr1</td>
<td>SR1 (not working well)</td>
</tr>
</tbody>
</table>

Default: bfgs

**line_search_method (advanced)**: Globalization method used in backtracking line search ←

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.
### Filter method

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>Filter method</td>
</tr>
<tr>
<td>cg-penalty</td>
<td>Chen-Goldfarb penalty function</td>
</tr>
<tr>
<td>penalty</td>
<td>Standard penalty function</td>
</tr>
</tbody>
</table>

Default: filter

#### linear_scaling_on_demand

Flag indicating that linear scaling is only done if it seems required.

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

Range: yes, no

Default: yes

#### linear_solver

Linear solver used for step computations.

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, that MA27, MA57, MA86, and MA97 are included with a commercially supported GAMS/IpoptH license only. To use MA27, MA57, MA86, or MA97 with GAMS/Ipopt, or to use HSL_MA77, a HSL library needs to be provided by the user. To use Pardiso from pardiso-project.org, a Pardiso library needs to be provided by the user. **ATTENTION**: Before Ipopt 3.14 (GAMS 36), value pardiso specified to use Pardiso from Intel MKL. With GAMS 36, this value has been renamed to pardisomkl.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma27</td>
<td>IpoptH: use the Harwell routine MA27; Ipopt: load the Harwell routine MA27 from user-provided library</td>
</tr>
<tr>
<td>ma57</td>
<td>IpoptH: use the Harwell routine MA57; Ipopt: load the Harwell routine MA57 from user-provided library</td>
</tr>
<tr>
<td>ma77</td>
<td>load the Harwell routine HSL_MA77 from user-provided library</td>
</tr>
<tr>
<td>ma86</td>
<td>IpoptH: use the Harwell routine HSL_MA86; Ipopt: load the Harwell routine HSL_MA86 from user-provided library</td>
</tr>
<tr>
<td>ma97</td>
<td>IpoptH: use the Harwell routine HSL_MA97; Ipopt: load the Harwell routine HSL_MA97 from user-provided library</td>
</tr>
<tr>
<td>pardiso</td>
<td>load the Pardiso package from pardiso-project.org from user-provided library at runtime</td>
</tr>
<tr>
<td>pardisomkl</td>
<td>use the Pardiso package from Intel MKL</td>
</tr>
<tr>
<td>mumps</td>
<td>use the Mumps package</td>
</tr>
</tbody>
</table>

Default: ma27, if IpoptH, otherwise mumps

#### linear_system_scaling

Method for scaling the linear system.

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. Note, that MC19 is included with a commercially supported GAMS/IpoptH license only. To use MC19 with GAMS/Ipopt, a HSL library needs to be provided by the user.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no scaling will be performed</td>
</tr>
<tr>
<td>mc19</td>
<td>IpoptH: use the Harwell routine MC19; Ipopt: load the Harwell routine MC19 from user-provided library</td>
</tr>
<tr>
<td>slack-based</td>
<td>use the slack values</td>
</tr>
</tbody>
</table>

Default: mc19, if IpoptH, otherwise none

**ma27_ignore_singularity (advanced)**: Whether to use MA27's ability to solve a linear system even if the matrix is singular.

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix.

Range: yes, no

Default: no

**ma27_la_init_factor**: Real workspace memory for MA27.

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by ma27_meminc_factor if required.

Range: [1, ∞]

Default: 5

**ma27_liw_init_factor**: Integer workspace memory for MA27.

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by ma27_meminc_factor if required.

Range: [1, ∞]

Default: 5

**ma27_meminc_factor**: Increment factor for workspace size for MA27.

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor.

Range: [1, ∞]

Default: 2

**ma27_pivtol**: Pivot tolerance for the linear solver MA27.

A smaller number pivots for sparsity, a larger number pivots for stability.

Range: (0, 1)

Default: 1e-08

**ma27_pivtolmax**: Maximum pivot tolerance for the linear solver MA27.
Ipopt may increase pivtol as high as ma27.pivtolmax to get a more accurate solution to the linear system.

Range: (0, 1)
Default: 0.0001

**ma27.print_level**: Debug printing level for the linear solver MA27

0: no printing; 1: Error messages only; 2: Error and warning messages; 3: Error and warning messages and terse monitoring; 4: All information.

Range: {0, ..., 4}
Default: 0

**ma27.skip_inertia_check** *(advanced)*: Whether to always pretend that inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no
Default: no

**ma57.automatic_scaling**: Controls whether to enable automatic scaling in MA57

For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

Range: yes, no
Default: no

**ma57.block_size**: Controls block size used by Level 3 BLAS in MA57BD

This is ICNTL(11) in MA57.

Range: {1, ..., ∞}
Default: 16

**ma57.node_amalgamation**: Node amalgamation parameter

This is ICNTL(12) in MA57.

Range: {1, ..., ∞}
Default: 16

**ma57.pivot_order**: Controls pivot order in MA57

This is ICNTL(6) in MA57.

Range: {0, ..., 5}
Default: 5

**ma57.pivtol**: Pivot tolerance for the linear solver MA57
A smaller number pivots for sparsity, a larger number pivots for stability.

Range: \((0, 1)\)
Default: \(1e-08\)

\textbf{ma57.pivtolmax}: Maximum pivot tolerance for the linear solver MA57.

Ipopt may increase pivtol as high as \texttt{ma57.pivtolmax} to get a more accurate solution to the linear system.

Range: \((0, 1)\)
Default: 0.0001

\textbf{ma57.pre alloc}: Safety factor for work space memory allocation for the linear solver MA57.

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice.

Range: \([1, \infty)\]
Default: 1.05

\textbf{ma57.print level}: Debug printing level for the linear solver MA57

0: no printing; 1: Error messages only; 2: Error and warning messages; 3: Error and warning messages and terse monitoring; \(\geq 4\): All information.

Range: \(\{0, ..., \infty\}\)
Default: 0

\textbf{ma57.small pivot flag}: Handling of small pivots

If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient. This is ICNTL(16) in MA57.

Range: \(\{0, ..., 1\}\)
Default: 0

\textbf{ma77.buffer.lpage}: Number of scalars per MA77 in-core buffer page in the out-of-core solver MA77

Must be at most \texttt{ma77.file.size}.

Range: \(\{1, ..., \infty\}\)
Default: 4096

\textbf{ma77.buffer.npage}: Number of pages that make up MA77 buffer

Number of pages of size \texttt{buffer.lpage} that exist in-core for the out-of-core solver MA77.

Range: \(\{1, ..., \infty\}\)
Default: 1600
**ma77_file_size**: Target size of each temporary file for MA77, scalars per type

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

Range: \{1, ..., \infty\}

Default: 2097152

**ma77_maxstore**: Maximum storage size for MA77 in-core mode

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

Range: \{0, ..., \infty\}

Default: 0

**ma77_nemin**: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.

Range: \{1, ..., \infty\}

Default: 8

**ma77_order**: Controls type of ordering used by MA77

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amd</td>
<td>Use the HSL_MC68 approximate minimum degree algorithm</td>
</tr>
<tr>
<td>metis</td>
<td>Use the MeTiS nested dissection algorithm (if available)</td>
</tr>
</tbody>
</table>

Default: metis

**ma77_print_level**: Debug printing level for the linear solver MA77

\(<0: \text{no printing}; 0: \text{Error and warning messages only}; 1: \text{Limited diagnostic printing}; >1: \text{Additional diagnostic printing.}

Range: \{-\infty, ..., \infty\}

Default: -1

**ma77_small**: Zero Pivot Threshold

Any pivot less than ma77_small is treated as zero.

Range: \[0, \infty\]

Default: 1e-20

**ma77_static**: Static Pivoting Threshold
See MA77 documentation. Either ma77_static=0.0 or ma77_static>ma77_small. ma77_static=0.0 disables static pivoting.

Range: [0, ∞]
Default: 0

**ma77_u:** Pivoting Threshold ↔

See MA77 documentation.

Range: [0, 0.5]
Default: 1e-08

**ma77_umax:** Maximum Pivoting Threshold ↔

Maximum value to which u will be increased to improve quality.

Range: [0, 0.5]
Default: 0.0001

**ma86_nemin:** Node Amalgamation parameter ↔

Two nodes in elimination tree are merged if result has fewer than ma86_nemin variables.

Range: {1, ..., ∞}
Default: 32

**ma86_order:** Controls type of ordering ↔

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Try both AMD and MeTiS, pick best</td>
</tr>
<tr>
<td>amd</td>
<td>Use the HSL_MC68 approximate minimum degree algorithm</td>
</tr>
<tr>
<td>metis</td>
<td>Use the MeTiS nested dissection algorithm (if available)</td>
</tr>
</tbody>
</table>

Default: auto

**ma86_print_level:** Debug printing level ↔

<0: no printing; 0: Error and warning messages only; 1: Limited diagnostic printing; >1 Additional diagnostic printing.

Range: {-∞, ..., ∞}
Default: -1

**ma86_scaling:** Controls scaling of matrix ↔

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Do not scale the linear system matrix</td>
</tr>
<tr>
<td>mc64</td>
<td>Scale linear system matrix using MC64</td>
</tr>
<tr>
<td>mc77</td>
<td>Scale linear system matrix using MC77 [1,3,0]</td>
</tr>
</tbody>
</table>
Default: mc64

**ma86_small**: Zero Pivot Threshold

Any pivot less than ma86_small is treated as zero.

Range: \([0, \infty]\)

Default: 1e-20

**ma86Static**: Static Pivoting Threshold

See MA86 documentation. Either ma86_static=0.0 or ma86_static>ma86_small. ma86_static=0.0 disables static pivoting.

Range: \([0, \infty]\)

Default: 0

**ma86_u**: Pivoting Threshold

See MA86 documentation.

Range: \([0, 0.5]\)

Default: 1e-08

**ma86_umax**: Maximum Pivoting Threshold

Maximum value to which u will be increased to improve quality.

Range: \([0, 0.5]\)

Default: 0.0001

**ma97_nemin**: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma97_nemin variables.

Range: \(\{1, ..., \infty\}\)

Default: 8

**ma97_order**: Controls type of ordering

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Use HSL_MA97 heuristic to guess best of AMD and METIS</td>
</tr>
<tr>
<td>best</td>
<td>Try both AMD and MeTiS, pick best</td>
</tr>
<tr>
<td>amd</td>
<td>Use the HSL_MC68 approximate minimum degree algorithm</td>
</tr>
<tr>
<td>metis</td>
<td>Use the MeTiS nested dissection algorithm</td>
</tr>
<tr>
<td>matched-auto</td>
<td>Use the HSL_MC80 matching with heuristic choice of AMD or METIS</td>
</tr>
<tr>
<td>matched-metis</td>
<td>Use the HSL_MC80 matching based ordering with METIS</td>
</tr>
<tr>
<td>matched-amd</td>
<td>Use the HSL_MC80 matching based ordering with AMD</td>
</tr>
</tbody>
</table>

Default: auto
**ma97_print_level**: Debug printing level

<0: no printing; 0: Error and warning messages only; 1: Limited diagnostic printing; >1 Additional diagnostic printing.

Range: \([-\infty, \ldots, \infty]\)

Default: -1

**ma97_scaling**: Specifies strategy for scaling

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Do not scale the linear system matrix</td>
</tr>
<tr>
<td>mc30</td>
<td>Scale all linear system matrices using MC30</td>
</tr>
<tr>
<td>mc64</td>
<td>Scale all linear system matrices using MC64</td>
</tr>
<tr>
<td>mc77</td>
<td>Scale all linear system matrices using MC77 ([1,3,0])</td>
</tr>
<tr>
<td>dynamic</td>
<td>Dynamically select scaling according to rules specified by ma97_scalingX and ma97_switchX options.</td>
</tr>
</tbody>
</table>

Default: dynamic

**ma97_scaling1** *(advanced)*: First scaling

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch1. If ma97_switch2 is triggered it is disabled.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No scaling</td>
</tr>
<tr>
<td>mc30</td>
<td>Scale linear system matrix using MC30</td>
</tr>
<tr>
<td>mc64</td>
<td>Scale linear system matrix using MC64</td>
</tr>
<tr>
<td>mc77</td>
<td>Scale linear system matrix using MC77 ([1,3,0])</td>
</tr>
</tbody>
</table>

Default: mc64

**ma97_scaling2** *(advanced)*: Second scaling

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch2. If ma97_switch3 is triggered it is disabled.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No scaling</td>
</tr>
<tr>
<td>mc30</td>
<td>Scale linear system matrix using MC30</td>
</tr>
<tr>
<td>mc64</td>
<td>Scale linear system matrix using MC64</td>
</tr>
<tr>
<td>mc77</td>
<td>Scale linear system matrix using MC77 ([1,3,0])</td>
</tr>
</tbody>
</table>

Default: mc64

**ma97_scaling3** *(advanced)*: Third scaling

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch3.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No scaling</td>
</tr>
<tr>
<td>mc30</td>
<td>Scale linear system matrix using MC30</td>
</tr>
<tr>
<td>mc64</td>
<td>Scale linear system matrix using MC64</td>
</tr>
<tr>
<td>mc77</td>
<td>Scale linear system matrix using MC77 [1,3,0]</td>
</tr>
</tbody>
</table>

Default: mc64

**ma97_small**: Zero Pivot Threshold

Any pivot less than ma97_small is treated as zero.

Range: \([0, \infty]\)

Default: 1e-20

**ma97_solve blas3 (advanced)**: Controls if blas2 or blas3 routines are used for solve

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Use BLAS2 (faster, some implementations bit incompatible)</td>
</tr>
<tr>
<td>yes</td>
<td>Use BLAS3 (slower)</td>
</tr>
</tbody>
</table>

Default: no

**ma97_switch1 (advanced)**: First switch, determine when ma97_scaling1 is enabled.

If ma97_scaling=dynamic, ma97_switch1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>never</td>
<td>Scaling is never enabled.</td>
</tr>
<tr>
<td>at_start</td>
<td>Scaling to be used from the very start.</td>
</tr>
<tr>
<td>at_start_reuse</td>
<td>Scaling to be used on first iteration, then reused thereafter.</td>
</tr>
<tr>
<td>on_demand</td>
<td>Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).</td>
</tr>
<tr>
<td>on_demand_reuse</td>
<td>As on_demand, but reuse scaling from previous itr</td>
</tr>
<tr>
<td>high_delay</td>
<td>Scaling to be used after more than 0.05*n delays are present</td>
</tr>
<tr>
<td>high_delay_reuse</td>
<td>Scaling to be used only when previous itr created more than 0.05*n additional delays, otherwise reuse scaling from previous itr</td>
</tr>
<tr>
<td>od_hd</td>
<td>Combination of on_demand and high_delay</td>
</tr>
<tr>
<td>od_hd_reuse</td>
<td>Combination of on_demand_reuse and high_delay_reuse</td>
</tr>
</tbody>
</table>

Default: od_hd_reuse

**ma97_switch2 (advanced)**: Second switch, determine when ma97_scaling2 is enabled.

If ma97_scaling=dynamic, ma97_switch2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>never</td>
<td>Scaling is never enabled.</td>
</tr>
<tr>
<td>at_start</td>
<td>Scaling to be used from the very start.</td>
</tr>
<tr>
<td>at_start_reuse</td>
<td>Scaling to be used on first iteration, then reused thereafter.</td>
</tr>
<tr>
<td>on_demand</td>
<td>Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).</td>
</tr>
<tr>
<td>on_demand_reuse</td>
<td>As on_demand, but reuse scaling from previous itr</td>
</tr>
<tr>
<td>high_delay</td>
<td>Scaling to be used after more than 0.05*n delays are present</td>
</tr>
<tr>
<td>high_delay_reuse</td>
<td>Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr</td>
</tr>
<tr>
<td>od_hd</td>
<td>Combination of on_demand and high_delay</td>
</tr>
<tr>
<td>od_hd_reuse</td>
<td>Combination of on_demand_reuse and high_delay_reuse</td>
</tr>
</tbody>
</table>

Default: never

**ma97_switch3** *(advanced)*: Third switch, determine when ma97_scaling3 is enabled. 

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>never</td>
<td>Scaling is never enabled.</td>
</tr>
<tr>
<td>at_start</td>
<td>Scaling to be used from the very start.</td>
</tr>
<tr>
<td>at_start_reuse</td>
<td>Scaling to be used on first iteration, then reused thereafter.</td>
</tr>
<tr>
<td>on_demand</td>
<td>Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).</td>
</tr>
<tr>
<td>on_demand_reuse</td>
<td>As on_demand, but reuse scaling from previous itr</td>
</tr>
<tr>
<td>high_delay</td>
<td>Scaling to be used after more than 0.05*n delays are present</td>
</tr>
<tr>
<td>high_delay_reuse</td>
<td>Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr</td>
</tr>
<tr>
<td>od_hd</td>
<td>Combination of on_demand and high_delay</td>
</tr>
<tr>
<td>od_hd_reuse</td>
<td>Combination of on_demand_reuse and high_delay_reuse</td>
</tr>
</tbody>
</table>

Default: never

**ma97_u**: Pivoting Threshold

See MA97 documentation.

Range: [0, 0.5]

Default: 1e-08

**ma97_umax**: Maximum Pivoting Threshold

See MA97 documentation.

Range: [0, 0.5]

Default: 0.0001

**max_cpu_time**: Maximum number of CPU seconds.
A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding message.

Range: \((0, \infty]\)
Default: \(1e+20\)

**max_filter_resets** *(advanced)*: Maximal allowed number of filter resets

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

Range: \(\{0, ..., \infty\}\)
Default: 5

**max_hessian_perturbation**: Maximum value of regularization parameter for handling negative curvature.

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. This is \(\delta_w^{\max}\) in the implementation paper.

Range: \((0, \infty]\)
Default: \(1e+20\)

**max_iter**: Maximum number of iterations.

The algorithm terminates with a message if the number of iterations exceeded this number.

Range: \(\{0, ..., \infty\}\)
Default: GAMS iterlim

**max_refinement_steps**: Maximum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

Range: \(\{0, ..., \infty\}\)
Default: 10

**max_resto_iter** *(advanced)*: Maximum number of successive iterations in restoration phase.

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

Range: \(\{0, ..., \infty\}\)
Default: 3000000
max_soc: Maximum number of second order correction trial steps at each iteration. 

Choosing 0 disables the second order corrections. This is p^{\text{max}} of Step A-5.9 of Algorithm A in the implementation paper.

Range: \{0, ..., \infty\}

Default: 4

max_soft_resto_iters (advanced): Maximum number of iterations performed successively in soft restoration phase.

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

Range: \{0, ..., \infty\}

Default: 10

max_wall_time: Maximum number of walltime clock seconds.

A limit on walltime clock seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding message.

Range: (0, \infty]

Default: GAMS reslim

mehrotra_algorithm: Indicates whether to do Mehrotra's predictor-corrector algorithm.

If enabled, line search is disabled and the (unglobalized) adaptive mu strategy is chosen with the "probing" oracle, and "corrector_type=affine" is used without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult". The Mehrotra's predictor-corrector algorithm works usually very well for LPs and convex QPs.

Range: yes, no

Default: no

min_hessian_perturbation: Smallest perturbation of the Hessian block.

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. This is delta_w^{\text{min}} in implementation paper.

Range: [0, \infty]

Default: 1e-20

min_refinement_steps: Minimum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

Range: \{0, ..., \infty\}

Default: 1

mu_allow_fast_monotone_decrease (advanced): Allow skipping of barrier problem if barrier test is already met.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Take at least one iteration per barrier problem even if the barrier test is already met for the updated barrier parameter</td>
</tr>
<tr>
<td>yes</td>
<td>Allow fast decrease of mu if barrier test it met</td>
</tr>
</tbody>
</table>

Default: yes

**mu_init**: Initial value for the barrier parameter. ←

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone")

Range: $$(0, \infty)$$

Default: 0.1

**mu_linear_decrease_factor**: Determines linear decrease rate of barrier parameter. ←

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of $\mu \times \text{mu_linear_decrease_factor}$ and $\mu^{\text{superlinear_decrease_power}}$. This is kappa_mu in implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: $$(0, 1)$$

Default: 0.2

**mu_max**: Maximum value for barrier parameter. ←

This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: $$(0, \infty)$$

Default: 100000

**mu_max_fact**: Factor for initialization of maximum value for barrier parameter. ←

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: $$(0, \infty)$$

Default: 1000

**mu_min**: Minimum value for barrier parameter. ←

This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of 1e-11 and min("tol","compl_inf_tol")/("barrier_tol_factor"+1), which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: $$(0, \infty)$$

Default: 1e-11

**mu_oracle**: Oracle for a new barrier parameter in the adaptive strategy. ←

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy").
5.20 IPOPT and IPOPTH

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>probing</td>
<td>Mehrotra's probing heuristic</td>
</tr>
<tr>
<td>logq</td>
<td>LOQO's centrality rule</td>
</tr>
<tr>
<td>quality-function</td>
<td>minimize a quality function</td>
</tr>
</tbody>
</table>

Default: quality-function

**mu_strategy**: Update strategy for barrier parameter. ➔

Determines which barrier parameter update strategy is to be used.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>monotone</td>
<td>use the monotone (Fiacco-McCormick) strategy</td>
</tr>
<tr>
<td>adaptive</td>
<td>use the adaptive update strategy</td>
</tr>
</tbody>
</table>

Default: adaptive

**mu_superlinear_decrease_power**: Determines superlinear decrease rate of barrier parameter. ➔

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of \( \mu^*\mu_{\text{linear\_decrease\_factor}} \) and \( \mu^\theta \mu_{\text{superlinear\_decrease\_power}} \). This is \( \theta \mu \) in implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: \((1, 2)\)

Default: 1.5

**mu_target**: Desired value of complementarity. ➔

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of “mu_target” specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

Range: \([0, \infty]\)

Default: 0

**mumps dep tol** *(advanced)*: Threshold to consider a pivot at zero in detection of linearly dependent constraints with MUMPS. ➔

This is CNTL(3) in MUMPS.

Range: real

Default: 0

**mumps_mem_percent**: Percentage increase in the estimated working space for MUMPS. ➔
When significant extra fill-in is caused by numerical pivoting, larger values of `mumps_mem_percent` may help use the workspace more efficiently. On the other hand, if memory requirements are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

Range: \( \{0, \ldots, \infty\} \)
Default: 1000

**mumps_permutingscaling:** Controls permuting and scaling in MUMPS

This is ICNTL(6) in MUMPS.

Range: \( \{0, \ldots, 7\} \)
Default: 7

**mumps_pivot_order:** Controls pivot order in MUMPS

This is ICNTL(7) in MUMPS.

Range: \( \{0, \ldots, 7\} \)
Default: 7

**mumps_pivtol:** Pivot tolerance for the linear solver MUMPS

A smaller number pivots for sparsity, a larger number pivots for stability.

Range: \([0, 1]\)
Default: 1e-06

**mumps_pivtolmax:** Maximum pivot tolerance for the linear solver MUMPS

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system.

Range: \([0, 1]\)
Default: 0.1

**mumps_print_level:** Debug printing level for the linear solver MUMPS

0: no printing; 1: Error messages only; 2: Error, warning, and main statistic messages; 3: Error and warning messages and terse diagnostics; \( \geq 4\): All information.

Range: \( \{0, \ldots, \infty\} \)
Default: 0

**mumps_scaling:** Controls scaling in MUMPS

This is ICNTL(8) in MUMPS.

Range: \([-2, \ldots, 77]\)
Default: 77

**neg_curv_test_reg:** Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014).
5.20 IPOPT and IPOPTH

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>use primal regularization with the inertia-free curvature test</td>
</tr>
<tr>
<td>no</td>
<td>use original IPOPT approach, in which the primal regularization is ignored</td>
</tr>
</tbody>
</table>

Default: yes

**neg_curv_test_tol**: Tolerance for heuristic to ignore wrong inertia. 

If nonzero, incorrect inertia in the augmented system is ignored, and Ipopt tests if the direction is a direction of positive curvature. This tolerance is alpha_n in the paper by Zavala and Chiang (2014) and it determines when the direction is considered to be sufficiently positive. A value in the range of [1e-12, 1e-11] is recommended.

Range: \([0, \infty]\)

Default: 0

**nlp_scaling_constr_target_gradient** *(advanced)*: Target value for constraint function gradient size.

If a positive number is chosen, the scaling factors for the constraint functions are computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

Range: \([0, \infty]\)

Default: 0

**nlp_scaling_max_gradient**: Maximum gradient after NLP scaling.

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is \(g_{\text{max}}\) in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Range: \((0, \infty]\)

Default: 100

**nlp_scaling_method**: Select the technique used for scaling the NLP.

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no problem scaling will be performed</td>
</tr>
<tr>
<td>gradient-based</td>
<td>scale the problem so the maximum gradient at the starting point is (nlp_scaling_max_gradient)</td>
</tr>
<tr>
<td>equilibration-based</td>
<td>scale the problem so that first derivatives are of order 1 at random points (GAMS/Ipopt: requires user-provided library with HSL routine MC19)</td>
</tr>
</tbody>
</table>

Default: gradient-based if GAMS scaleopt is not set, otherwise none

**nlp_scaling_min_value**: Minimum value of gradient-based scaling values.
This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Range: [0, \infty]
Default: 1e-08

\textbf{nlp\_scaling\_obj\_target\_gradient (advanced):} Target value for objective function gradient size. 

If a positive number is chosen, the scaling factor for the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides \texttt{nlp\_scaling\_max\_gradient} for the objective function.

Range: [0, \infty]
Default: 0

\textbf{nu\_inc (advanced):} Increment of the penalty parameter.

Range: (0, \infty]
Default: 0.0001

\textbf{nu\_init (advanced):} Initial value of the penalty parameter.

Range: (0, \infty]
Default: 1e-06

\textbf{obj\_max\_inc (advanced):} Determines the upper bound on the acceptable increase of barrier objective function.

Trial points are rejected if they lead to an increase in the barrier objective function by more than \texttt{obj\_max\_inc} orders of magnitude.

Range: (1, \infty]
Default: 5

\textbf{pardiso\_iter\_coarse\_size (advanced):} Maximum Size of Coarse Grid Matrix

DPARM(3)
Range: \{1, ..., \infty\}
Default: 5000

\textbf{pardiso\_iter\_dropping\_factor (advanced):} dropping value for incomplete factor

DPARM(5)
Range: (0, 1)
Default: 0.5

\textbf{pardiso\_iter\_dropping\_schur (advanced):} dropping value for sparsify schur complement factor
DPARM(6)
Range: (0, 1)
Default: 0.1

**pardiso_iter_inverse_norm_factor** *(advanced):* \( \leftarrow \)

DPARM(8)
Range: \((1, \infty]\)
Default: 5e+06

**pardiso_iter_max_levels** *(advanced):* Maximum Size of Grid Levels \( \leftarrow \)

DPARM(4)
Range: \(\{1, \ldots, \infty\}\)
Default: 10

**pardiso_iter_max_row_fill** *(advanced):* max fill for each row \( \leftarrow \)

DPARM(7)
Range: \(\{1, \ldots, \infty\}\)
Default: 10000000

**pardiso_iter_relative_tol** *(advanced):* Relative Residual Convergence \( \leftarrow \)

DPARM(2)
Range: (0, 1)
Default: 1e-06

**pardiso_iterative** *(advanced):* Switch for iterative solver in Pardiso library \( \leftarrow \)

Range: yes, no
Default: no

**pardiso_matching_strategy**:
Matching strategy to be used by Pardiso \( \leftarrow \)

This is IPAR(13) in Pardiso manual.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete</td>
<td>Match complete (IPAR(13)=1)</td>
</tr>
<tr>
<td>complete+2x2</td>
<td>Match complete+2x2 (IPAR(13)=2)</td>
</tr>
<tr>
<td>constraints</td>
<td>Match constraints (IPAR(13)=3)</td>
</tr>
</tbody>
</table>

Default: complete+2x2
**pardiso_max_droptol_corrections** *(advanced):* Maximal number of decreases of drop tolerance during one solve.  

This is relevant only for iterative Pardiso options.

Range: \{1, ..., ∞\}

Default: 4

**pardiso_max_iter** *(advanced):* Maximum number of Krylov-Subspace Iteration  

DPARM(1)

Range: \{1, ..., ∞\}

Default: 500

**pardiso_max_iterative_refinement_steps**: Limit on number of iterative refinement steps.  

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: \{-∞, ..., ∞\}

Default: 0

**pardiso_msglvl**: Pardiso message level  

This is MSGlvl in the Pardiso manual.

Range: \{0, ..., ∞\}

Default: 0

**pardiso_order**: Controls the fill-in reduction ordering algorithm for the input matrix.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amd</td>
<td>minimum degree algorithm</td>
</tr>
<tr>
<td>one</td>
<td></td>
</tr>
<tr>
<td>metis</td>
<td>MeTiS nested dissection algorithm</td>
</tr>
<tr>
<td>pmetis</td>
<td>parallel (OpenMP) version of MeTiS nested dissection algorithm</td>
</tr>
<tr>
<td>four</td>
<td></td>
</tr>
<tr>
<td>five</td>
<td></td>
</tr>
</tbody>
</table>

Default: metis

**pardiso_redo_symbolic_fact_only_if_inertia_wrong** *(advanced):* Toggle for handling case when elements were perturbed by Pardiso.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Always redo symbolic factorization when elements were perturbed</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Only redo symbolic factorization when elements were perturbed if also the inertia was wrong</td>
</tr>
</tbody>
</table>

Default: no

**pardiso_repeated_perturbation_means_singular** *(advanced):* Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization. 

Range: yes, no

Default: no

**pardiso_skip_inertia_check** *(advanced):* Whether to pretend that inertia is correct. 

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no

Default: no

**pardisolib:** Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime

Range: string

Default: libpardiso.so (Linux), libpardiso.dylib (macOS), libpardiso.dll (Windows)

**pardismkl_matching_strategy:** Matching strategy to be used by Pardiso

This is IPAR(13) in Pardiso manual.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete</td>
<td>Match complete (IPAR(13)=1)</td>
</tr>
<tr>
<td>complete+2x2</td>
<td>Match complete+2x2 (IPAR(13)=2)</td>
</tr>
<tr>
<td>constraints</td>
<td>Match constraints (IPAR(13)=3)</td>
</tr>
</tbody>
</table>

Default: complete+2x2

**pardismkl_max_iterative_refinement_steps:** Limit on number of iterative refinement steps.

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: \(-\infty, ..., \infty\)

Default: 1

**pardismkl_msglvl:** Pardiso message level
This is MSGLVL in the Pardiso manual.

Range: \( \{0, \ldots, \infty\} \)

Default: 0

**pardisomkl_order**: Controls the fill-in reduction ordering algorithm for the input matrix.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amd</td>
<td>minimum degree algorithm</td>
</tr>
<tr>
<td>one</td>
<td>undocumented</td>
</tr>
<tr>
<td>metis</td>
<td>MeTiS nested dissection algorithm</td>
</tr>
<tr>
<td>pmetis</td>
<td>parallel (OpenMP) version of MeTiS nested dissection algorithm</td>
</tr>
</tbody>
</table>

Default: metis

**pardisomkl_redo_symbolic_fact_only_if_inertia_wrong (advanced)**: Toggle for handling case when elements were perturbed by Pardiso.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Always redo symbolic factorization when elements were perturbed</td>
</tr>
<tr>
<td>yes</td>
<td>Only redo symbolic factorization when elements were perturbed if also the inertia was wrong</td>
</tr>
</tbody>
</table>

Default: no

**pardisomkl_repeated_perturbation_means_singular (advanced)**: Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization.

Range: yes, no

Default: no

**pardisomkl_skip_inertia_check (advanced)**: Whether to pretend that inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no

Default: no

**perturb_always_cd (advanced)**: Active permanent perturbation of constraint linearization.

Enabling this option leads to using the \( \delta_c \) and \( \delta_d \) perturbation for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

Range: yes, no

Default: no

**perturb_dec_fact**: Decrease factor for x-s perturbation.
The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. This is kappa_w^- in the implementation paper.

Range: (0, 1)
Default: 0.333333

perturb_inc_fact: Increase factor for x-s perturbation. ←

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. This is kappa_w^+ in the implementation paper.

Range: (1, ∞]
Default: 8

perturb_inc_fact_first: Increase factor for x-s perturbation for very first perturbation. ←

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for the first perturbation than that used for the remaining perturbations. This is bar_kappa_w^+ in the implementation paper.

Range: (1, ∞]
Default: 100

print_advanced_options (advanced): whether to print also advanced options ←

Range: yes, no
Default: no

print_eval_error: Switch to enable printing information about function evaluation errors into the GAMS listing file. ←

Range: no, yes
Default: yes

print_frequency_iter: Determines at which iteration frequency the summarizing iteration output line should be printed. ←

Summarizing iteration output is printed every print_frequency_iter iterations, if at least print_frequency_time seconds have passed since last output.

Range: \{1, ..., ∞\}
Default: 1

print_frequency_time: Determines at which time frequency the summarizing iteration output line should be printed. ←

Summarizing iteration output is printed if at least print_frequency_time seconds have passed since last output and the iteration number is a multiple of print_frequency_iter.

Range: [0, ∞]
Default: 0
**print_info_string**: Enables printing of additional info string at end of iteration output.  

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.

Range: yes, no

Default: no

**print_level**: Output verbosity level.  

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: \{0, ..., 12\}

Default: 5

**print_options_mode**: format in which to print options documentation  

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Ordinary text</td>
</tr>
<tr>
<td>latex</td>
<td>LaTeX formatted</td>
</tr>
<tr>
<td>doxygen</td>
<td>Doxygen (markdown) formatted</td>
</tr>
</tbody>
</table>

Default: text

**print_timing_statistics**: Switch to print timing statistics.  

If selected, the program will print the time spend for selected tasks. This implies timing_statistics=yes.

Range: yes, no

Default: no

**quality_function_balancing_term** *(advanced)*: The balancing term included in the quality function for centrality.  

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. Only used if option "mu_oracle" is set to "quality-function".

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no balancing term is added</td>
</tr>
<tr>
<td>cubic</td>
<td>$\text{Max}(0,\text{Max}(\text{dual inf, primal inf})-\text{compl})^3$</td>
</tr>
</tbody>
</table>

Default: none

**quality_function_centrality** *(advanced)*: The penalty term for centrality that is included in quality function.  

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the $\text{x}_i$ in the Loqo update rule. Only used if option "mu_oracle" is set to "quality-function".

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no balancing term is added</td>
</tr>
<tr>
<td>cubic</td>
<td>$\text{Max}(0,\text{Max}(\text{dual inf, primal inf})-\text{compl})^3$</td>
</tr>
</tbody>
</table>

Default: none
5.20 IPOPT and IPOPTH

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no penalty term is added</td>
</tr>
<tr>
<td>log</td>
<td>complementarity * the log of the centrality measure</td>
</tr>
<tr>
<td>reciprocal</td>
<td>complementarity * the reciprocal of the centrality measure</td>
</tr>
<tr>
<td>cubed-reciprocal</td>
<td>complementarity * the reciprocal of the centrality measure cubed</td>
</tr>
</tbody>
</table>

Default: none

**quality_function_max_section_steps**: Maximum number of search steps during direct search procedure determining the optimal centering parameter. 

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: \{0, ..., \infty\}

Default: 8

**quality_function_norm_type (advanced)**: Norm used for components of the quality function.

Only used if option "mu_oracle" is set to "quality-function".

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-norm</td>
<td>use the 1-norm (abs sum)</td>
</tr>
<tr>
<td>2-norm-squared</td>
<td>use the 2-norm squared (sum of squares)</td>
</tr>
<tr>
<td>max-norm</td>
<td>use the infinity norm (max)</td>
</tr>
<tr>
<td>2-norm</td>
<td>use 2-norm</td>
</tr>
</tbody>
</table>

Default: 2-norm-squared

**quality_function_section_qf_tol (advanced)**: Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: [0, 1)

Default: 0

**quality_function_section_sigma_tol (advanced)**: Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: [0, 1)

Default: 0.01

**recale_y**: Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than reccale_y_feas_tol. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>use the Newton step to update the multipliers</td>
</tr>
<tr>
<td>yes</td>
<td>use least-square multiplier estimates</td>
</tr>
</tbody>
</table>

Default: no

**recale_y_feas_tol**: Feasibility threshold for recomputation of multipliers.  

If recalc_y is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

Range: $(0, \infty)$

Default: $1e-06$

**replace_bounds** *(advanced)*: Whether all variable bounds should be replaced by inequality constraints

This option must be set for the inexact algorithm.

Range: yes, no

Default: no

**report_mininfeas_solution**: Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.

This option allows to obtain the most feasible solution found by Ipopt during the iteration process, if it stops at a (locally) infeasible solution, due to a limit (time, iterations, ...), or with a failure in the restoration phase.

Range: no, yes

Default: no

**required_infeasibility_reduction**: Required reduction of infeasibility before leaving restoration phase.

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: $[0, 1)$

Default: 0.9

**residual_improvement_factor** *(advanced)*: Minimal required reduction of residual test ratio in iterative refinement.

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

Range: $(0, \infty]$  

Default: 1

**residual_ratio_max** *(advanced)*: Iterative refinement tolerance
Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

Range: \((0, \infty]\)

Default: 1e-10

**residual_ratio_singular** *(advanced)*: Threshold for declaring linear system singular after failed iterative refinement. ⇐

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

Range: \((0, \infty]\)

Default: 1e-05

**resto_failure_feasibility_threshold** *(advanced)*: Threshold for primal infeasibility to declare failure of restoration phase. ⇐

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is actually \(1e2 \times \text{tol}\), where \(\text{tol}\) is the general termination tolerance.

Range: \([0, \infty]\)

Default: 0

**resto_penalty_parameter** *(advanced)*: Penalty parameter in the restoration phase objective function. ⇐

This is the parameter \(\rho\) in equation (31a) in the Ipopt implementation paper.

Range: \((0, \infty]\)

Default: 1000

**resto_proximity_weight** *(advanced)*: Weighting factor for the proximity term in restoration phase objective. ⇐

This determines how the parameter \(\zeta\) in equation (29a) in the implementation paper is computed. \(\zeta\) here is \(\text{resto_proximity_weight} \times \sqrt{\mu}\), where \(\mu\) is the current barrier parameter.

Range: \([0, \infty]\)

Default: 1

**rho** *(advanced)*: Value in penalty parameter update formula. ⇐

Range: \((0, 1]\)

Default: 0.1

**s_max** *(advanced)*: Scaling threshold for the NLP error. ⇐
See paragraph after Eqn. (6) in the implementation paper.

Range: \((0, \infty]\)

Default: 100

**s_phi** *(advanced) :* Exponent for linear barrier function model in the switching rule.

See Eqn. (19) in the implementation paper.

Range: \((1, \infty]\)

Default: 2.3

**s_theta** *(advanced) :* Exponent for current constraint violation in the switching rule.

See Eqn. (19) in the implementation paper.

Range: \((1, \infty]\)

Default: 1.1

**sigma_max** *(advanced) :* Maximum value of the centering parameter.

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. Only used if option "mu_oracle" is set to "quality-function".

Range: \((0, \infty]\)

Default: 100

**sigma_min** *(advanced) :* Minimum value of the centering parameter.

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. Only used if option "mu_oracle" is set to "quality-function".

Range: \([0, \infty]\)

Default: 1e-06

**skip_corr_if_neg_curv** *(advanced) :* Whether to skip the corrector step in negative curvature iteration.

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Range: yes, no

Default: yes

**skip_corr_in_monotone_mode** *(advanced) :* Whether to skip the corrector step during monotone barrier parameter mode.

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Range: yes, no

Default: yes
**slack_bound_frac**: Desired minimum relative distance from the initial slack to bound.  

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)  

Range: (0, 0.5]  
Default: 0.01

**slack_bound_push**: Desired minimum absolute distance from the initial slack to bound.  

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)  

Range: (0, \infty]  
Default: 0.01

**slack_move** *(advanced)*: Correction size for very small slacks.  

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is `mach_\text{eps}^{3/4}`. See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.  

Range: [0, \infty]  
Default: 1.81899e-12

**soc_method**: Ways to apply second order correction  

This option determines the way to apply second order correction. 0 is the method described in the implementation paper. 1 is the modified way which adds alpha on the rhs of x and s rows.  

Range: \{0, ..., 1\}  
Default: 0

**soft_resto_pdeerror_reduction_factor**: Required reduction in primal-dual error in the soft restoration phase.  

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing “0” here disables the soft restoration phase.  

Range: [0, \infty]  
Default: 0.9999

**start_with_resto**: Whether to switch to restoration phase in first iteration.  

Setting this option to “yes” forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.  

Range: yes, no  
Default: no
**tau_min** *(advanced)*: Lower bound on fraction-to-the-boundary parameter tau. 

This is tau_min in the implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: (0, 1)
Default: 0.99

**theta_max_fact** *(advanced)*: Determines upper bound for constraint violation in the filter. 

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

Range: (0, \( \infty \)]
Default: 10000

**theta_min_fact** *(advanced)*: Determines constraint violation threshold in the switching rule. 

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

Range: (0, \( \infty \)]
Default: 0.0001

**timing_statistics**: Indicates whether to measure time spend in components of Ipopt and NLP evaluation.

The overall algorithm time is unaffected by this option.

Range: yes, no
Default: no

**tiny_step_tol** *(advanced)*: Tolerance for detecting numerically insignificant steps. 

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

Range: [0, \( \infty \)]
Default: 2.22045e-15

**tiny_step_y_tol** *(advanced)*: Tolerance for quitting because of numerically insignificant steps. 

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

Range: [0, \( \infty \)]
Default: 0.01
5.20 IPOPT and IPOPTH

**tol**: Desired convergence tolerance (relative).

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. This is epsilon_tol in Eqn. (6) in implementation paper. See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Range: $(0, \infty]$

Default: $1e^{-08}$

**warm_start_bound_frac**: same as bound_frac for the regular initializer

Range: $(0, 0.5]$.

Default: $0.001$

**warm_start_bound_push**: same as bound_push for the regular initializer

Range: $(0, \infty]$

Default: $0.001$

**warm_start_init_point**: Warm-start for initial point

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>do not use the warm start initialization</td>
</tr>
<tr>
<td>yes</td>
<td>use the warm start initialization</td>
</tr>
</tbody>
</table>

Default: yes, if run on modified model instance (e.g., from GUSS), otherwise no

**warm_start_mult_bound_push**: same as mult_bound_push for the regular initializer

Range: $(0, \infty]$

Default: $0.001$

**warm_start_mult_init_max**: Maximum initial value for the equality multipliers.

Range: real

Default: $1e+06$

**warm_start_slack_bound_frac**: same as slack_bound_frac for the regular initializer

Range: $(0, 0.5]$

Default: $0.001$

**warm_start_slack_bound_push**: same as slack_bound_push for the regular initializer
Range: \((0, \infty]\)
Default: 0.001

**warm_start_target_mu (advanced):** \(\leftarrow\)

Experimental!

Range: real
Default: 0

**watchdog_shortened_iter_trigger:** Number of shortened iterations that trigger the watchdog. \(\leftarrow\)

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

Range: \(\{0, \ldots, \infty\}\)
Default: 10

**watchdog_trial_iter_max:** Maximum number of watchdog iterations. \(\leftarrow\)

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

Range: \(\{1, \ldots, \infty\}\)
Default: 3

### 5.21 JAMS and LogMIP

#### 5.21.1 Introduction

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematical programming reformulations. The idea behind EMP is that new upcoming types of models which currently cannot be solved reliably are reformulated into models of established math programming classes in order to use mature solver technology. At this stage, EMP supports the modeling of Bilevel Programs, Variational Inequalities, Disjunctive Programs, Extended Nonlinear Programs and Embedded Complementarity Systems.

Extended mathematical programs are collections of functions and variables joined together using specific optimization and complementarity primitives. EMP annotates the existing relationships within a model to facilitate higher level structure identification. A specific implementation of this framework is outlined that reformulates the original GAMS model automatically using directives contained in an "empinfo" file into an equivalent model that can be solved using existing GAMS solvers.

The reformulation is done by the solver JAMS which currently is the only solver that is capable of handling EMP models. Examples showing how to use the EMP framework and the solver JAMS are made available through the GAMS EMP Library which is included in the GAMS Distribution. In order to generate a copy of and EMPLIB model, one can use the library facility of the GAMS IDE, or execute the command line directive `emplib <modelname>`, where modelname is the (stem of the) file containing the model.

EMP has been developed jointly by Michael Ferris of UW-Madison, Ignacio Grossmann of Carnegie Mellon University, and GAMS Development Corporation. EMP and JAMS come free of charge with any licensed GAMS system but require a subsolver to solve the generated models.
5.21.2 JAMS: a reformulation tool

EMP models are currently processed by the JAMS solver. The solver JAMS creates a scalar version of the given GAMS model. This scalar version of the model is then solved by an appropriate subsolver. By default, there are no reformulations carried out, so the model generated is simply a GAMS scalar form of the model the actual subsolver will process. The subsolver used is by default the currently specified solver for the given model type.

5.21.2.1 The JAMS Option File

As with any GAMS solver, JAMS has an option file, typically called jams.opt. A JAMS option subsolver is available to change the subsolver used for the reformulated model, along with an option to utilize a subsolver option file (subsolveropt).

The actual scalar version of the model can also be seen by the modeler using the option filename. For example, the option file

```
subsolver path
subsolveropt 1
filename mcpmod.gms
```

when applied to an EMP model that is a complementarity problem will create a file called mcpmod.gms in the current directory and solve that model using the solver PATH utilizing any options for PATH that are specified in path.opt. The scalarized model is not particularly useful to look at since all of the original variables have been renamed into a scalar form. The mapping between original variables and the ones used in the scalar version of the model is given in a dictionary file that can also be seen by the modeler using the dict option. If a user simply wants to generate this scalar model, then the option terminate will not solve the generated model.

After the scalar version of the model is solved, the solution values are mapped back into the original namespace and returned to the modeler as usual in the listing file. The JAMS option margtol allows the modeler to suppress reporting marginals that have (absolute) values smaller than this tolerance.

Obviously, all of the above functionality is not of much value: the key part of JAMS is to interpret additional directives to take the original model and produce a reformulated scalar model. This is carried out using an “empinfo” file. The syntax and use of this file is the content of the remaining sections of this document.

The option EMPInfoFile allows the user to specify the path and name of a file containing additional EMP information. The syntax of this file is described by examples elsewhere in this document and in section Empinfo file details. There is a subtlety that should be mentioned when a user writes this file in the default location in the GAMS scratch directory of the current run specified in the gams file using:

```
file empinfo / '%emp.info%' /;
```

wherein certain additional formatting instructions are given. If instead, the file handle is given by:

```
file empinfo / 'empinfo.txt' /;
```

for example, then to produce the same file the following lines must be added to the gams source file:

```
empinfo.pc = 8;
empinfo.pw = 255;
```

The following tables list all available options.

5.21.2.2 Reformulation Options
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeepObj</td>
<td>Keep original objective function and variables in generated MCP model. This option is only valid for reformulations into complementarity systems. If KeepObj is set the generated MCP program will incorporate the original objective function and the Karush-Kuhn-Tucker condition of the original objective variable.</td>
<td>0</td>
</tr>
<tr>
<td>MargTol</td>
<td>Only report marginals with an absolute value above the tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>ObjVarName</td>
<td>Name of objective variable in generated model.</td>
<td>objvar</td>
</tr>
</tbody>
</table>

#### 5.21.2.3 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompModel</td>
<td>Complementarity model (MCP or VI) to generate for equilibrium problems.</td>
<td>mcp</td>
</tr>
<tr>
<td>Dict</td>
<td>Generate a dictionary file that maps original variable and equation names to the ones of the scalar JAMS model.</td>
<td>dict.txt</td>
</tr>
<tr>
<td>DisjBinRelax</td>
<td>Relax requirement that disjunction variables are binary.</td>
<td>0</td>
</tr>
<tr>
<td>EMPInfoFile</td>
<td>Path and name of file containing additional EMP information as disjunctions, perpendiculares, bilevel characterization etc. By default, the file is expected to exist in the GAMS scratch directory of the current run (gams.scrdir). If this option is used, EMP searches for the file based on the provided relative or absolute path. In case of a given relative path, it is assumed to be relative to the working directory.</td>
<td></td>
</tr>
<tr>
<td>FileName</td>
<td>Filename of generated scalar reformulated GAMS model.</td>
<td>emp.gms</td>
</tr>
<tr>
<td>ImplVarModel</td>
<td>Reformulation model to generate for implicit variables.</td>
<td>switching</td>
</tr>
<tr>
<td>NLConsToFunc</td>
<td>Stick nonlinear constraints into functional part when generating a VI model for equilibrium problems</td>
<td>0</td>
</tr>
<tr>
<td>SharedEqu</td>
<td>Allow shared equations in equilibrium problems</td>
<td>0</td>
</tr>
<tr>
<td>SubSolvePar</td>
<td>User defined GAMS parameters for subsolve</td>
<td></td>
</tr>
<tr>
<td>SubSolver</td>
<td>Subsolver used to solve the reformulated model. The subsolver chosen has to be suitable for the chosen reformulation type. The user can also provide a solver related option file, see SubSolverOpt.</td>
<td></td>
</tr>
<tr>
<td>SubSolverOpt</td>
<td>Optfile value to pass to the subsolver</td>
<td>1</td>
</tr>
<tr>
<td>Terminate</td>
<td>Generate the GAMS source code of the reformulated model in a file and terminate without solving the model.</td>
<td>0</td>
</tr>
<tr>
<td>UserPFFile</td>
<td>Filename of extra GAMS options when running the reformulated GAMS model.</td>
<td></td>
</tr>
<tr>
<td>ZipDebug</td>
<td>Zip-file name to create if the internal GAMS model fails</td>
<td>none</td>
</tr>
</tbody>
</table>
5.21.3 Forming Optimality Conditions: NLP2MCP

The first nontrivial use of the JAMS solver is to automatically generate the first order conditions of a linear or nonlinear program; essentially we reformulate the optimization problem as a mixed complementarity problem (MCP). The "empinfo" file to do this simply contains the following line:

```
modeltype mcp
```

Behind the scenes, JAMS forms the Lagrangian of the nonlinear program and then forms its Karush-Kuhn-Tucker optimality conditions. To be clear, given the original nonlinear program

\[
\begin{align*}
\min_x f(x) \text{ s.t. } & g(x) \leq 0, \ h(x) = 0, \\
\end{align*}
\]

the Lagrangian is:

\[
L(x, \lambda, \mu) = f(x) - \langle \lambda, g(x) \rangle - \langle \mu, h(x) \rangle.
\]

The first order conditions are the following MCP:

\[
\begin{align*}
0 &= \nabla_x L(x, \lambda, \mu) \perp x \text{ free} \\
0 &\geq -\nabla_\lambda L(x, \lambda, \mu) \perp \lambda \leq 0 \\
0 &= -\nabla_\mu L(x, \lambda, \mu) \perp \mu \text{ free}
\end{align*}
\]

A specific example is:

\[
\begin{align*}
\min_{x,y,z} & -3x + y \\
\text{s.t.} & \quad x + y \leq 1, \ x + y - z = 2, \ x, y \geq 0
\end{align*}
\]

which is found in the EMPLIB model `nlp2mcp`:

Positive Variables x, y;
Variables f, z;
Equations g, h, defobj;
\[
\begin{align*}
g.. & \quad x + y =e= 1; \\
h.. & \quad x + y - z =e= 2; \\
defobj.. & \quad f =e= -3x + y; \end{align*}
\]

model comp / defobj, g, h /;
file info / '%emp.info%' /;
putclose info / 'modeltype mcp';
solve comp using emp minimizing f;

The putclose line writes out the default "empinfo" file whose location is provided in the system string `%emp.info%`. Armed with this additional information, the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 &\leq -3 - \lambda - \mu \perp x \geq 0 \\
0 &\leq 1 - \lambda - \mu \perp y \geq 0 \\
0 &= \mu \perp z \text{ free} \\
0 &\geq x + y - 1 \perp \lambda \leq 0 \\
0 &= x + y - z - 2 \perp \mu \text{ free}.
\end{align*}
\]
5.21.4 Soft Constraints

In many cases, we wish to relax certain constraints in a model during solution (to help identify feasibility issues for example). As an example, consider the problem

\[
\begin{align*}
\min_{x_1, x_2, x_3} & \quad \exp(x_1) \\
\text{s.t.} & \quad \log(x_1) = 1, \\
& \quad x_2^2 \leq 2, \\
& \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0,
\end{align*}
\]

which can be formulated in GAMS as

```gams
$title simple example of ENLP
Variables obj, x1, x2, x3;
Equations f0, f1, f2, f3, f4;
f0.. obj =e= exp(x1);
f1.. log(x1) =e= 1;
f2.. sqrt(x2) =g= 2;
f3.. x1/x2 =e= log(x3);
f4.. 3*x1 + x2 =l= 5;
x1.lo = 0; x2.lo = 0;
model enlpemp /all/;
x1.l = 1; x2.l = 1; x3.l = 1;
solve enlpemp using nlp min obj;
```

5.21.4.1 Reformulation as a classical NLP

Soft constraints allow us to treat certain equations in the model as "soft" by removing the constraints and adding a penalty term to the objective function. Explicitly, we replace the above problem by:

\[
\begin{align*}
\min_{x_1, x_2, x_3, w} & \quad \exp(x_1) + 5 \|\log(x_1) - 1\|^2 + 2 \max(x_2^2 - 2, 0) \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0.
\end{align*}
\]

In this problem, we still force \( x_1/x_2 = \log(x_3) \), but apply a least squares penalty to \( \log(x_1) - 1 \) and a smaller one-sided penalization to \( x_2^2 - 2 \).

The above formulation is nonsmooth due to the "max" term in the objective function; in practice we would replace this by:

\[
\begin{align*}
\min_{x_1, x_2, x_3, w} & \quad \exp(x_1) + 5 (\log(x_1) - 1)^2 + 2w \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\
& \quad w \geq x_2^2 - 2, w \geq 0
\end{align*}
\]

and recover a standard form NLP.

The "empinfo" file:

```gams
modeltype NLP
adjustequ f1 sqr 5
f2 maxz 2
```

coupled with replacing the last line with

```gams
solve enlpemp using emp min obj;
```

achieves this goal. The parameter values provide the penalty coefficients above.
5.21 JAMS and LogMIP

5.21.4.2 Reformulation as an MCP

As an alternative, we can rewrite the problem as an MCP, also dealing explicitly with the nonsmoothness. The "empinfo" file is given by:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

and this generates the following MCP:

\[
\begin{align*}
0 &= \log(x_1) - 1 + y_1/10 \quad \perp \quad y_1 \text{ free}, \\
0 &\leq x_2^2 - 2 \quad \perp \quad y_2 \geq 0, \\
0 &= x_1/x_2 - \log(x_3) \quad \perp \quad y_3 \text{ free}, \\
0 &\geq 3x_1 + x_2 - 5 \quad \perp \quad y_4 \leq 0, \\
0 &\leq \exp(x_1) - y_1/x_1 - y_3/x_2 - 3y_4 \quad \perp \quad x_1 \geq 0, \\
0 &\leq -2y_2x_2 + x_1y_3/x_2^2 - y_4 \quad \perp \quad x_2 \geq 0, \\
0 &= y_3/x_3 \quad \perp \quad x_3 \text{ free},
\end{align*}
\]

where \( y \) represent the multipliers.

A complete description of the process to derive this MCP will be given later in section Extended Nonlinear Programs.

5.21.5 Bilevel Programs

Mathematical programs with optimization problems in their constraints have a long history in operations research including [15] [29] [71]. New codes are being developed that exploit this structure, at least for simple hierarchies, and attempt to define and implement algorithms for their solution.

The simplest case is that of bilevel programming, where an upper level problem depends on the solution of a lower level optimization. For example:

\[
\begin{align*}
\min_{x,y} & \quad f(x,y) \\
\text{s.t.} & \quad g(x,y) \leq 0, \\
& \quad y \text{ solves } \min_y v(x,y) \text{ s.t. } h(x,y) \geq 0.
\end{align*}
\]

Often, the upper level is referred to as the "leader", while the lower level is the "follower".

This problem can be reformulated as a Mathematical Program with Complementarity Constraints (MPCC) by replacing the lower level optimization problem by its first order optimality conditions:

\[
\begin{align*}
\min_{x,y} & \quad f(x,y) \\
\text{s.t.} & \quad g(x,y) \leq 0, \\
0 &= \nabla_y v(x,y) - \lambda^T \nabla_y h(x,y) \quad \perp \quad x \text{ free} \\
0 &\leq h(x,y) \quad \perp \quad \lambda \geq 0.
\end{align*}
\]

We find a solution of the MPCC, not of the bilevel program. This approach allows the MPCC to be solved using the NLPEC code, for example. Note that this reformulation is potentially problematic. First order
conditions require theoretical assumptions to be necessary and sufficient for local optimality. There may be cases where the lower level problem has multiple local solutions, but the modeler really was interested in the global solution. The approach here may not produce this solution, even if a global solver is used within NLPEC.

The following example is Example 5.1.1, page 197 from [15]. Mathematically, the problem is

$$\begin{align*}
\min_{x,y} & \quad x - 4y \\
\text{s.t.} & \quad y \text{ solves } \min y \\
& \quad x + y \geq 3 \\
& \quad 2x - y \geq 0 \\
& \quad -2x - y \geq -12 \\
& \quad -3x + 2y \geq -4
\end{align*}$$

and the EMPLIB model bard511 contains the following code:

```plaintext
Positive variables x, y;
Variables objout, objin;
equations defout, defin, e1, e2, e3, e4;
defout.. objout =e= x - 4*y;
defin.. objin =e= y;
e1.. x + y =g= 3;
e2.. 2*x - y =g= 0;
e3.. -2*x - y =g= -12;
e4.. -3*x + 2*y =g= -4;
model bard / all /;
$echo bilevel x min objin y defin e1 e2 e3 e4 > "%emp.info%"
solve bard using emp minimizing objout;
```

Note that first the functions that form the objectives and constraints of the model are defined and the model is assembled. The `$echo` line writes the "empinfo" file and states that the lower level problem involves the objective `objin` which is to be minimized by choice of variables `y` subject to the constraints specified in (`defin`), `e1`, `e2`, `e3`, and `e4`.

Note that the variables `x` are declared to be variables of the upper level problem and this example has no upper level constraints `g`. Having written the problem in this way, the MPCC is generated automatically, and passed on to a solver. In the case where that solver is NLPEC, a further reformulation of the model is carried out to convert the MPCC into an equivalent NLP or a parametric sequence of NLP's.

Further examples of bilevel models in EMPLIB are named: bard*, ccmg74, ccmg153, flds*, jointc1, jointc2, mirrlees, transbp.

The EMP model type allows multiple lower level problems to be specified within the bilevel format. An example of this is given in EMPLIB as ccmg71. The equations and objectives are specified in the normal manner; the only change is the definition of the "empinfo" file, shown below:

```plaintext
... defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4);
e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6;
...
$echo > "%emp.info%"
bilevel x1 x2 x3 x4
min h1 u1 u2 u3 u4 defh1 e1
min h2 v1 v2 v3 v4 defh2 e2
$offecho
```

This corresponds to a bilevel program with two followers, both solving minimization problems. The first follower minimizes the objective function `h1` (defined in `defh1`) over the variables `u1`, `u2`, `u3`, and `u4` subject to the constraint given in `e1`. The second followers problem is defined analogously next. Note that `h1` involves the variables `x1`, `x2`, `x3`, and `x4` that are optimization variables of the leader. The constraint in `e1` could also include these variables, and also the variables `v1`, `v2`, `v3`, or `v4` of the second follower, but all of these would be treated as parameters by the first follower.

The actual model (ccmg71) in EMPLIB uses a shortcut notation:

```plaintext
... defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4);
```

Further examples of bilevel models in EMPLIB are named: bard*, ccmg74, ccmg153, flds*, jointc1, jointc2, mirrlees, transbp.
In the followers problem, the 'w' notation indicates that this agent will optimize over all the variables used in defh1 and e1 that are not under the control of any other follower or the leader. In this case, this means u1, u2, u3, and u4. To avoid confusion, it is recommended that the modeler explicitly names all the variables in each followers problem as shown before.

5.21.6 Variational Inequalities

A variational inequality $VI(F,X)$ is to find $x \in X$:

$$\langle F(x), (z - x) \rangle \geq 0, \text{ for all } z \in X.$$ 

Here $X$ is a closed (frequently assumed convex) set, defined for example as

$$X = \{ x \mid x \geq 0, h(x) \geq 0 \}.$$ 

Note that the first-order (minimum principle) conditions of a nonlinear program

$$\min_{z \in X} f(z)$$

are precisely of this form with $F(x) = \nabla f(x)$.

It is well known that such problems can be reformulated as complementarity problems when the set $X$ has the representation

$$X = \{ x \mid x \geq 0, h(x) \geq 0 \}$$

by introducing multipliers $\lambda$ on the constraints $h(x) \geq 0:

$$0 \leq F(x) - \langle \lambda, \nabla h(x) \rangle \perp x \geq 0$$

$$0 \leq h(x) \perp \lambda \geq 0.$$ 

If $X$ has a different representation, this construction would be modified appropriately.

A simple two dimensional example may be useful to improve understanding. Let

$$F(x) = \begin{bmatrix} x_1 + 2 \\ x_1 + x_2 - 3 \end{bmatrix}, \quad X = \{ x \geq 0 \mid x_1 + x_2 \leq 1 \},$$

so that $F$ is an affine function, but $F$ is not the gradient of any function $f : \mathbb{R}^2 \to \mathbb{R}$. For this particular data, $VI(F,X)$ has a unique solution $x = (0, 1)$.

Set J / 1, 2 /;
Positive Variable x(J) 'vars, perp to f(J)';
Equations F(J), h;
F(J).. (x('1') + 2)*$sameas(J,'1') + (x('1') + x('2') - 3)*$sameas(J,'2') =n= 0 ;
h.. x('1') + x('2') =l= 1;
model simpleVI / F, h/;
pUTCLOSE fx 'vi F x h';
solve simpleVI using emp;

Note that the first lines of this file define the $F$ and $h$ using standard GAMS syntax and include the defining equations in the model simpleVI. The extension is the annotation "empinfo" file that indicates certain equations are to be treated differently by the EMP tool. The annotation simply says that the model is a VI (vi) that pairs F with x and that the remaining (unpaired) equations form the constraint set X. (Alternative notation allows the keyword vi to be replaced by vifunc.) Thus, model equations F define a function F that is to be part of a variational inequality, while the equations h define constraints of X. It is also acceptable in this setting to use the "empinfo" file defined by:
In this case, by default, any equations that are given in the model statement but not included as a pair in the \textit{vi} statement are automatically used to form $X$. An alternative way to write this model without using \texttt{sameas} is given in \texttt{EMPLIB} as \texttt{affinevi}.

Some subtleties related to VI's are demonstrated in the \texttt{EMPLIB} model \texttt{zerofunc}. In this model, the set $X$ is defined using variables $y$ and $z$, for which $z$ does not appear in the definition of $F \equiv F_y$. In this case, the variable $z$ is then matched with a "0" function. The "empinfo" file can be written in a number of different ways:

\texttt{putclose fx 'vi F_y y';}

or

\texttt{putclose fx 'vi z F_y y';}

or

\texttt{putclose fx 'vi z F_y y gCons';}

or

\texttt{putclose fx 'vi F_z z F_y y gCons';}

where $F_z$ simply defines the zero function. To some extent, our preferred notation is the one listed third: it explicitly includes all the variables and constraints that are present in the model and does not require the modeler to define $F_z$ at all.

Further example models in \texttt{EMPLIB} are \texttt{simplevi}, \texttt{simplevi2}, \texttt{simplevi3}, \texttt{target}, \texttt{traffic}, \texttt{traffic2}, and \texttt{transvi}.

Note also that the lower level problems of a bilevel program could be VI's instead of optimization problems - these problems are called Mathematical Programs with Equilibrium Constraints (MPEC) in the literature. Note that since MPCC is a special case of MPEC, the GAMS model type MPEC covers both. An example demonstrating this setup is given in \texttt{EMPLIB} as \texttt{multmpec}. The actual model to solve is:

$$\begin{align*}
\min_{z, u, v, w} & z \\
\text{s.t.} & \exp(z) + w = 2, z \geq 1 \\
& (u, v) \text{ solves } VI([v + w + z - 1; u - \log(v)], \{(u, v) \mid u \geq 0, v \geq 0\}) \\
& w \text{ solves } VI(w + z + 3, \{w \mid w \text{ free}\})
\end{align*}$$

Note that the two VI's (due to the definitional sets) correspond respectively to a complementarity problem:

$$
\begin{align*}
0 \leq v + w + z - 1 & \perp u \geq 0 \\
0 \leq u - \log(v) & \perp v \geq 0
\end{align*}
$$

and a linear equation:

$$w + z + 3 = 0$$
The actual GAMS code is as follows:

Positive Variable u;
Variables v, w, z;
Equations f1, f2, f3, h;
f1.. v + w + z =n= 1;
f2.. u =n= log(v);
f3.. w + z =n= -3;
h.. exp(z) + w =e= 2;
v.lo = 0; v.l = 1; z.lo = 1;
model mpecmod /all/;
$onecho > %emp.info%
bilevel
vi f1 u
f2 v
vi f3 w
$offecho
solve mpecmod using emp min z;

The initial level value for v \((v.l = 1)\) is needed to protect the evaluation of log(v). The two complementarity problems are specified in the empinfo file (it is not really necessary to split these VI's apart, but it may give information to a solver that can be exploited). It is of course possible to write the MPCC directly in GAMS using the model type MPEC instead of EMP.

### 5.21.7 Embedded Complementarity Systems

A different type of embedded optimization model that arises frequently in applications is:

\[
\begin{align*}
\max \quad & f(x, y) \\
\text{s.t.} \quad & g(x, y) \leq 0 \quad (\perp p \geq 0) \\
& H(x, y, p) = 0 \quad (\perp y \text{ free})
\end{align*}
\]

Note the difference here: the optimization problem is over the variable \(x\), and is parameterized by the variable \(y\). The choice of \(y\) is fixed by the (auxiliary) complementarity relationships depicted here by \(H\). Note that the \(H\) equations are not part of the optimization problem, but are essentially auxiliary constraints to tie down remaining variables in the model.

A specific example is:

\[
\begin{align*}
\max \quad & x \\
\text{s.t.} \quad & x + y \leq 1 \\
& -3x + y = 0.5 \quad (\perp y \text{ free})
\end{align*}
\]

which is found in the EMPLIB model \texttt{simpequil2}: Variables y;
Positive variables x;
Equations optcons, vicons;
optcons.. x + y =l= 1;
vicons.. -3*x + y =e= 0.5;
model comp / optcons, vicons /;
file info / '%emp.info%' /;
put info / 'equilibrium';
put / 'max x optcons';
putclose / 'vi vicons y';
solve comp using emp;

In order that this model can be processed correctly as an EMP, the modeler provides additional annotations to the model defining equations in an "empinfo" file. Specifically, first it is indicated that the problem is an equilibrium problem involving one or more agent problems. Next, the first agent is defined as an optimizer (over \(x\)). Finally, the second agent is defined as solving a VI in \(y\). Armed with this additional information, the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 \leq -1 + p & \perp x \geq 0 \\
0 \leq 1 - x - y & \perp p \geq 0 \\
0 = -3x + y - 0.5 & \perp y \text{ free},
\end{align*}
\]
(which is formed by the steps we outline below). EMP explicitly enforces the rule that every variable and constraint is under the control of exactly one agent. Thus a constraint or a variable cannot appear in both the max problem and the VI problem.

The above example is slightly simpler than the general form described above in which \( H \) is a function of \( x, y, \) and \( p, \) the multiplier on the constraint of the optimization problem. The problem is that we do not have that variable around in the model code if we only specify the optimization problem there. This occurs for example in the classical PIES Model due to Hogan. In this setting, the problem is described by a linear program

\[
\begin{align*}
\min_{x} & \quad c^T x \\
\text{s.t.} & \quad Ax = q(p) \\
& \quad Bx = b \\
& \quad x \geq 0
\end{align*}
\]

in which the quantity \( q \) is a function of \( p, \) which is a multiplier on one of the LP constraints. To do this in EMP, we simply add the annotation:

```plaintext
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'emp.info'/;
put myinfo 'equilibrium';
put 'min obj c o ct ot lt ht defobj dembal cmbal ombal lmbal hmbal ruse';
putclose 'dualvar p dembal';
solve piesemp using emp;
```

where `dembal` is the name of the constraint for which \( p \) needs to be the multiplier. The full model is found in the EMPLIB model `pies`. Two final points: the `dualvar` directive identifies the variable \( p \) with the multiplier on the `dembal` constraint, and all variables and constraints must be owned by a single agent. In this case, since there is only one agent (the minimizer), all constraints of the model are explicitly claimed, along with all variables except for \( p. \) However, next, \( p \) is identified with the `dembal` constraint, which is owned by the min agent, and hence \( p \) is also owned by that agent.

There are several shorthands possible here. The first is that the explicit statement of variables can be replaced by the `'*'` form:

```plaintext
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'emp.info'/;
put myinfo 'equilibrium';
put 'min obj * defobj dembal cmbal ombal lmbal hmbal ruse';
```

Alternatively, an even shorter version is possible since there is only one agent present in this model, namely:

```plaintext
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'emp.info'/;
putclose myinfo 'dualvar p dembal';
solve piesemp using emp minimizing obj;
```

Note that in this form, all the variables and constraints of the original model are included in the (single) agents problem, and the original variable \( p \) is identified in the constructed MCP with the multiplier on the `dembal` constraint.

In the general case where the "empinfo" file contains all three lines:

```plaintext
min x optcons
vi vicons y
dualvar p optcons
```

namely that the function \( H \) that is defined in `vicons` is complementary to the variable \( y \) (and hence the variable \( y \) is a parameter to the optimization problem), and furthermore that the dual variable associated with the equation `optcons` in the optimization problem is one and the same as the variable \( p \) used to define \( H, \) the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 &= \nabla_x L(x, y, p) \quad \perp x \text{ free} \\
0 &\geq -\nabla_p L(x, y, p) \quad \perp p \leq 0 \\
0 &= H(x, y, p) \quad \perp y \text{ free},
\end{align*}
\]
where the Lagrangian is defined as
\[ L(x, y, p) = f(x, y) - \langle p, g(x, y) \rangle. \]

Essentially, this MCP consists of the first order optimality conditions of the optimization problem, coupled with the VI that is the second agents problem. An example that does both of these things together is provided in EMPLIB as \texttt{scarfemp-primal}.

Note that since the PIES model has no \( y \) variables, this is a special case of the general form in which the second agents (VI) problem is simply not present.

Example models are \texttt{ferris43}, \texttt{flipper pies}, \texttt{scarfemp-dual}, \texttt{simpequil}, \texttt{transecs}, and \texttt{transeql}.

### 5.21.8 MOPECs

MOPECs (Multiple Optimization Problems with Equilibrium Constraints) are a simple conceptual extension of the aforementioned embedded complementarity system. Instead of having a single optimizing agent and a variational inequality, one instead allows multiple optimizing agents coupled with an equilibrium constraint (the variational inequality).

Perhaps the most popular use of this formulation is where competition is allowed between agents. A standard method to deal with such cases is via the concept of Nash Games. In this setting \( x^* \) is a Nash Equilibrium if
\[ x_i^* \in \arg \min_{x_i \in X_i} \ell_i(x_i, x_{-i}^*, q), \forall i \in I, \]
where \( x_{-i} \) are other players decisions and the quantities \( q \) are given exogenously, or via complementarity:
\[ 0 \leq H(x, q) \perp q \geq 0. \]

This mechanism is extremely popular in economics, and Nash famously won the Nobel Prize for his contributions to this literature.

This format is again an EMP, more general than the example given above in two respects. Firstly, there is more than one optimization problem specified in the embedded complementarity system. Secondly, the parameters in each optimization problem consist of two types. Firstly, there are the variables \( q \) that are tied down by the auxiliary complementarity condition and hence are treated as parameters by the \( i \)th Nash player. Also there are the variables \( x_{-i} \) that are treated as parameters by the \( i \)th Nash player, but are treated as variables by a different player \( j \).

While we do not specify the syntax here for these issues, \texttt{EMPLIB} provides examples that outline how to carry out this matching within GAMS. Finally, two points of note: first it is clear that the resulting model is a complementarity problem and can be solved using PATH, for example. Secondly, performing the conversion from an embedded complementarity system or a Nash Game automatically is a critical step in making such models practically useful.

We note that there is a large literature on discrete-time finite-state stochastic games: this has become a central tool in analysis of strategic interactions among forward-looking players in dynamic environments. The model of dynamic competition in an oligopolistic industry given in \cite{55} is exactly in the format described above, and has been used extensively in applications such as advertising, collusion, mergers, technology adoption, international trade and finance. Ongoing work aims to use the EMP format to model these problems.
5.21.9 Extended Nonlinear Programs

Optimization models have traditionally been of the form (NLP). Specialized codes have allowed certain problem structures to be exploited algorithmically, for example simple bounds on variables. However, for the most part, assumptions of smoothness of \( f, g \) and \( h \) are required for many solvers to process these problems effectively. In a series of papers, Rockafellar and colleagues [152] [153] [154] have introduced the notion of extended nonlinear programming, where the (primal) problem has the form:

\[
\min_{x \in X} f(x) + \theta(g_1(x), \ldots, g_m(x)).
\]

In this setting, \( X \) is assumed to be a nonempty polyhedral set, and the functions \( f, g_1, \ldots, g_m \) are smooth. The function \( \theta \) can be thought of as a generalized penalty function that may well be nonsmooth. However, when \( \theta \) has the form

\[
\theta(u) = \sup_{y \in Y} \{ \langle y, u \rangle - k(y) \},
\]

a computationally exploitable and theoretically powerful framework can be developed based on conjugate duality. A key point for computation and modeling is that the function \( \theta \) can be fully described by defining the set \( Y \) and the function \( k \). Furthermore, from a modeling perspective, an extended nonlinear program can be specified simply by defining the functions \( f, g_1, \ldots, g_m \) in the manner already provided by the modeling system, with the additional issue of simply defining \( Y \) and \( k \). Conceptually, this is not much harder than what is carried out already, but leads to significant enhancements to the types of models that are available. Once a modeler determines which constraints are treated via which choice of \( k \) and \( Y \), the EMP model interface automatically forms an equivalent variational inequality or complementarity problem. As we show later, there may be alternative formulations that are computationally more appealing; such reformulations can be generated using different options to JAMS.

5.21.9.1 Forms of \( \theta \)

The EMP model type makes the problem format (ENLP) available to users in GAMS. As special cases, we can model piecewise linear penalties, least squares, and \( L_1 \) approximation problems, as well as the notion of soft and hard constraints.

For ease of exposition, we now describe a subset of the types of functions \( \theta \) that can be generated by particular choices of \( Y \) and \( k \). In many cases, the function \( \theta \) is separable, that is

\[
\theta(u) = \sum_{i=1}^{m} \theta_i(u_i),
\]

so we can either specify \( \theta_i \) or \( \theta \) itself.

Extended nonlinear programs include the classical nonlinear programming form (NLP) as a special case. This follows from the observation that if \( K \) is a closed convex cone, and we let \( \psi_K \) denote the “indicator function” of \( K \) defined by:

\[
\psi_K(u) = \begin{cases} 0 & \text{if } u \in K, \\ \infty & \text{else,} \end{cases}
\]

then (NLP) can be rewritten as:

\[
\min_x f(x) + \psi_K((g(x), h(x)), \quad K = \mathbb{R}_-^m \times \{0\}^p,
\]

where \( m \) and \( p \) are the dimensions of \( g \) and \( h \) respectively and \( \mathbb{R}_-^m = \{ u \in \mathbb{R}^m \mid u \leq 0 \} \). An elementary calculation shows that

\[
\psi_K(u) = \sup_{v \in \mathbb{R}^p} \langle u, v \rangle,
\]
where $K^\circ = \{ u \mid \langle u, v \rangle \leq 0, \forall v \in K \}$ is the polar cone of the given cone $K$. Thus, when $\theta(u) = \psi_K(u)$ we simply take $k \equiv 0$ and $Y = K^\circ$. In our example, $K^\circ = \mathbb{R}^m_+ \times \mathbb{R}^p$. To some extent, this is just a formalism that allows us to claim the classical case as a specialization; however when we take the cone $K$ to be more general than the polyhedral cone used above, we can generate conic programs for example.

The second example involves a piecewise linear function $\theta$: Formally, for $u \in \mathbb{R}$,

$$\theta(u) = \begin{cases} \rho u & \text{if } u \geq 0, \\ \sigma u & \text{else.} \end{cases}$$

In this case, simple calculations prove that $\theta$ has the form $(\theta \text{ con})$ for the choices $k \equiv 0$ and $Y = [\sigma, \rho]$.

The special case where $\sigma = -\rho$ results in

$$\theta(u) = \rho |u| .$$

This allows us to model nonsmooth $L_1$ approximation problems. Another special case results from the choice of $\sigma = -\gamma$, $\rho = 0$, whereby

$$\theta(u) = \gamma \max\{-u, 0\} .$$

This formulation corresponds to a soft penalization on an inequality constraint, namely if $\theta(-g_1(x))$ is used then nothing is added to the objective function if $g_1(x) \leq 0$, but $\gamma g_1(x)$ is added if the constraint $g_1(x) \leq 0$ is violated. Contrast this to the classical setting above, where $\infty$ is added to the objective if the inequality constraint is violated. It is interesting to see that truncating the set $Y$, which amounts to bounding the multipliers, results in replacing the classical constraint by a linearized penalty.

The third example involves a more interesting choice of $k$. If we wish to replace the "absolute value" penalization given above by a quadratic penalization (as in classical least squares analysis), that is

$$\theta(u) = \gamma u^2$$

then a simple calculation shows that we should take

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = \mathbb{R} .$$

By simply specifying this different choice of $k$ and $Y$ we can generate such models easily and quickly within the modeling system. Note, however, that the reformulation we would use in $\theta(u) = \rho |u|$ and $\theta(u) = \gamma u^2$ are very different as we shall explain in the simple example below. Furthermore, in many applications it has become popular to penalize violations using a quadratic penalty only within a certain interval, afterwards switching to a linear penalization (chosen to make the penalty function $\theta$ continuously differentiable - see [99] ). That is:

$$\theta(u) = \begin{cases} \gamma u - \frac{1}{2} \gamma^2 & \text{if } u \geq \gamma \\ \frac{1}{2} u^2 & \text{if } u \in [-\gamma, \gamma] \\ -\gamma u + \frac{1}{2} \gamma^2 & \text{else.} \end{cases}$$

Such functions arise from quadratic $k$ and simple bound sets $Y$. In particular, the somewhat more general function

$$\theta(u) = \begin{cases} \gamma \beta^2 + \rho (u - \beta) & \text{if } u \geq \beta \\ \gamma u^2 & \text{if } u \in [\alpha, \beta] \\ \gamma \alpha^2 + \sigma (u - \alpha) & \text{else} \end{cases}$$

arises from the choice of

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = [\sigma, \rho] ,$$

with $\alpha = \frac{\sigma}{2\gamma}$ and $\beta = \frac{\rho}{2\gamma}$.

The final example that we give is that of $L_\infty$ penalization. This example is different to the examples given above in that $\theta$ is not separable. However, straightforward calculation can be used to show

$$\theta(u) = \max_{i=1,\ldots,m} u_i$$

results from the choice of

$$k \equiv 0 \text{ and } Y = \left\{ y \in \mathbb{R}^m \mid y \geq 0, \sum_{i=1}^m y_i = 1 \right\} ,$$

that is, $Y$ is the unit simplex.
5.21.9.2 Underlying theory

The underlying structure of \( \theta \) leads to a set of extended optimality conditions and an elegant duality theory. This is based on an extended form of the Lagrangian:

\[
L(x, y) = f(x) + \sum_{i=1}^{m} y_{i} g_{i}(x) - k(y)
\]

\[x \in X, y \in Y\]

Note that the Lagrangian \( L \) is smooth - all the nonsmoothness is captured in the \( \theta \) function. The theory is an elegant combination of calculus arguments related to \( g_{i} \) and its derivatives, and variational analysis for features related to \( \theta \).

It is shown in [153] that under a standard constraint qualification, the first-order conditions of (ENLP) are precisely in the form of the following variational inequality:

\[
\text{VI} \left( \begin{bmatrix} \nabla_x L(x, y) \\ -\nabla_y L(x, y) \end{bmatrix}, X \times Y \right). \tag{ENLP VI}
\]

When \( X \) and \( Y \) are simple bound sets, this is simply a complementarity problem.

Note that EMP exploits this result. In particular, if an extended nonlinear program of the form (ENLP) is given to EMP, then the optimality conditions (ENLP VI) are formed as a variational inequality problem and can be processed as outlined above. For a specific example, we cite the fact that if we use the (classical) choice \( k \equiv 0 \) and \( Y = K^{\circ} \), then the optimality conditions of (ENLP) are precisely the standard complementarity problem given as (ENLP VI). While this is of interest, we believe that other choices of \( k \) and \( Y \) may be more useful and lead to models that have more practical significance.

Under appropriate convexity assumptions on this Lagrangian, it can be shown that a solution of the VI (ENLP VI) is a saddle point for the Lagrangian on \( X \times Y \). Furthermore, in this setting, the saddle point generates solutions to the primal problem (ENLP) and its dual problem:

\[
\max_{y \in Y} d(y), \quad \text{where} \quad d(y) = \inf_{x \in X} L(x, y),
\]

with no duality gap.

Unfortunately, the perturbations \( y \) in Rockafellar's theory are precisely the negative of those used throughout the GAMS system. Thus, we need to replace \( y \) by \(-y\) throughout in the above to recover the same multipliers as those GAMS uses.

5.21.9.3 A simple example

As an example, consider the problem

\[
\min_{x_1, x_2, x_3} \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2\max(x_2^2 - 2, 0)
\]
\[\text{s.t. } x_1/x_2 = \log(x_3),
\]
\[3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0.\]

In this problem, we would take

\[
X = \{ x \in \mathbb{R}^3 \mid 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \}.
\]
The function $\theta$ essentially treats 3 separable pieces:

$$g_1(x) = \log(x_1) - 1,$$
$$g_2(x) = x_2^2 - 2,$$
$$g_3(x) = x_1/x_2 - \log(x_3).$$

A classical problem would force $g_1(x) = 0$, $g_2(x) \leq 0$, and $g_3(x) = 0$, while minimizing $f(x) = \exp(x_1)$. In our problem, we still force $g_3(x) = 0$, but apply a (soft) least squares penalty on $g_1(x)$ and a smaller one-sided penalization on $g_2(x)$. The above formulation is nonsmooth due to the "max" term in the objective function; in practice we could replace this by:

$$\min_{x_1,x_2,x_3,w} \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2w$$

s.t. $x_1/x_2 = \log(x_3)$,

$$3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0$$

$$w \geq x_2^2 - 2, w \geq 0$$

and recover a standard form NLP. If the penalty on $g_1(x)$ would be replaced by a one-norm penalization (instead of least squares), we would have to play a similar game, moving the function $g_1(x)$ into the constraints and adding additional variable(s). To some extent, this seems unnatural - a modeler should be able to interchange the penalization without having to reformulate the problem from scratch. The proposed extended NLP would not be reformulated at all by the modeler, but allows all these "generalized constraints" to be treated in a similar manner within the modeling system. The actual formulation would take:

$$\theta(u) = \theta_1(u_1) + \theta_2(u_2) + \theta_3(u_3)$$

where

$$\theta_1(u_1) = 5u_1^2,$$
$$\theta_2(u_2) = 2 \max(u_2, 0),$$
$$\theta_3(u_3) = \psi_{[0]}(u_3).$$

The discussion above allows us to see that

$$Y = -(\mathbb{R} \times [0, 2] \times \mathbb{R}),$$
$$k(y) = \frac{1}{20}y_1^2 + 0 + 0.$$

The corresponding Lagrangian is the smooth function:

$$\mathcal{L}(x, y) = f(x) - \sum_{i=1}^{3} y_i g_i(x) - k(y).$$

The corresponding VI (ENLP VI) can almost be formulated in GAMS (except that the linear constraint in $X$ cannot be handled currently except by introducing a $\theta_4(x)$). Thus

$$g_4(x) = 3x_1 + x_2 - 5, \theta_4(u) = \psi_{\mathbb{R}_-},$$

resulting in the following choices for $Y$ and $k$:

$$Y = -(\mathbb{R} \times [0, 2] \times \mathbb{R} \times \mathbb{R}_+),$$
$$k(y) = \frac{1}{20}y_1^2 + 0 + 0 + 0.$$

Since $X$ and $Y$ are now simple bound sets, (ENLP VI) is now a complementarity problem and can be solved for example using PATH. A simple "empinfo" file details the choices of $Y$ and $k$ from the implemented library:

```
Adjustequ
e1 sqr 5
e2 MaxZ 2
```
5.21.9.4 Reformulation as a classical NLP

Suppose

$$\theta(u) = \sup_{y \in Y} \{ \langle u, y \rangle - \frac{1}{2} \langle y, Qy \rangle \}$$

for a polyhedral set $Y \subseteq \mathbb{R}^m$ and a symmetric positive semidefinite $Q \in \mathbb{R}^{m \times m}$ (possibly $Q = 0$).

Suppose further that

$$X = \{ x \mid Rx \leq r \}, Y = \{ y \mid S^T y \leq s \},$$

$$Q = DJ^{-1}D^T, \quad F(x) = (g_1(x), \ldots, g_m(x)),$$

where $J$ is symmetric and positive definite (for instance $J = I$). Then, as outlined by [154], the optimal solutions $\bar{x}$ of (ENLP) are the $\bar{x}$ components of the optimal solutions $(\bar{x}, \bar{z}, \bar{w})$ to

$$\min f(x) + \langle s, z \rangle + \frac{1}{2} \langle w, Jw \rangle$$

s.t. $Rx \leq r, z \geq 0, F(x) - Sz - Dw = 0.$

The multiplier on the equality constraint in the usual sense is the multiplier associated with $\bar{x}$ in the extended Lagrangian for (ENLP). (Note that a Cholesky factorization may be needed to determine $D$.)

It may be better to solve this reformulated NLP than to solve the VI (ENLP VI). However, it is important that we can convey all types of nonsmooth optimization problems to a solver as smooth optimization problems, and hence it is important to communicate the appropriate structure to the solver interface. We believe that specifying $Y$ and $k$ is a theoretically sound way to do this.

5.21.10 Disjunctive Programs (LogMIP)

There are many ways that the EMP model type can be used for further extensions to the modeling capabilities of a given system. In particular, the procedures outlined in [190] for disjunctive programming extensions are also implemented within the EMP model type.

The disjunctive programming procedures are also known as the LogMIP 2.0 solver for solving linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for modeling discrete choices. While the modeling and solution of these disjunctive optimization problems has not yet reached the stage of maturity and reliability as LP, MIP and NLP modeling, these problems have a rich area of applications.

LogMIP 2.0 has been developed by Dr. Aldo Vecchietti from INGAR (Santa Fe, Argentina) and Professor Ignacio E. Grossmann from Carnegie Mellon University (Pittsburgh, USA), and supersedes its previous version, LogMIP 1.0 (GAMS releases 22.6 (December 2007) to 23.6 (December 2010)). Changes in version 2.0 are at the level of language, where now the EMP syntax and modeltype is used, and at the level of solvers, where Big-M and convex-hull relaxations are combined. For more information see the website http://www.logmip.ceride.gov.ar/ and the documentation http://www.logmip.ceride.gov.ar/files/pdfs/newUserManual.pdf.

One simple example to highlight the disjunctive programming feature is the notion of an ordering of tasks, namely that either job $i$ completes before job $j$ starts or the converse, i.e., that the jobs cannot overlap. Such a disjunction can be specified using an empinfo file containing lines:

```
disjunction * seq(i,j) else seq(j,i)
```

In such an example, one can implement a Big-M method, employ indicator constraints, or utilize a convex hull reformulation. The convex hull reformulation is the default strategy; to utilize the Big-M formulation, the additional option
default bigm 1000

would add binary variables and constraints to impose the disjunction using a Big-M value of 1000. Alternatively, the option setting

default indic

writes out a model and an option file that implements a reformulation using indicator constraints, which can be handled by the solvers CPLEX, SCIP, and XPRESS. The EMPLIB model sequence is sequencing model that implements all of these options.

More complicated (nonlinear) examples make the utility of this approach clearer. The design of a multiproduct batch plan with intermediate storage described in [189] and a synthesis problem involving 8 processes from [186] are also included in the EMP model library. As a final example, the gasoline emission model outlined in [72] is precisely in the form that could exploit the features of EMP related to (nonlinear) disjunctive programming.

5.21.11 Empinfo file details

We have explained above many of the standard vectorized ways to write an "empinfo" file. The "empinfo" file has a vectorized format and a more powerful (but more complex) scalar version. We describe some of the features of the scalar version in this section.

An example of the use of the scalar syntax is given below

```plaintext
file info / '%emp.info%' /;
pit info / 'equilibrium';
loop(h,
   put / 'min' obj(h) /;
   loop(j, put x(h,j)); put /;
   loop(k, put z(h,k));
   put / costdef(h) /;
   loop(i, put nodebal(h,i)); put /;
);
loop(a, put 'vi',flowdef(a),f(a) /);
```

This is an example of a MOPEC in which the optimization agents are indexed by $h$ and each of these agents control variables $x_{hj}$ and $z_{hk}$. The objective function for each $h$ is defined in `costdef`, the constraints of each minimization problem are defined in `nodebal`, and the equilibrium constraints that link all the optimization problems are a VI defined by variables $f_a$ and functions within `flowdef`. Sometimes it is necessary to use syntax that generates the "tl" fields of sets, but this seems only necessary when abnormal side conditions are needed.

The format of the empinfo file is given below:

```
Disjunction [chull [big eps] | bigM [big eps threshold] | indic]
   [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]
Default [chull [big eps] | bigM [big eps threshold] | indic]
ParallelStep1 {equ|*}
AdjustEqu equ abs|sqr|maxz|... {weight}
```
5.22 KESTREL - Remote Solver Execution on NEOS Servers

5.22.1 Background

The Kestrel client/server is a way of sending your GAMS solve job to be solved via the NEOS Server from within your usual GAMS modeling environment and receiving results that can be processed as with any local solver. Starting with distribution 23.7 the Kestrel solver is part of the GAMS distribution. The solve statement using the GAMS/KESTREL solver invokes a client program that sends your problem to a solver running on one of the NEOS Server's remote computers. The results from the NEOS Server are eventually returned through Kestrel to GAMS, where you can view and manipulate them locally in the usual way. The introduction below covers everything you need to know to start using GAMS/KESTREL. Further information about more advanced features and other uses of Kestrel can be found at the Kestrel page on the NEOS Server. Please also make you familiar with the NEOS optimization service terms of use before submitting jobs via GAMS/KESTREL.

5.22.1.1 Kestrel and IBM DOcloud

With GAMS 24.6 the GAMS/Kestrel client has been adjusted to work with IBM's cloud service Decision Optimization on Cloud (DOcloud). When the Kestrel option docloudurl is set, the generated model instance will be automatically translated into an MPS format and send to DOcloud and solved by Cplex in IBM's cloud. So no GAMS/Cplex license is required on your local machine. You as a user need to register with IBM on the web page mentioned about and pay IBM for the solution time spend in IBM's cloud. IBM provides details how to get started with DOcloud.

5.22.2 Using GAMS/KESTREL

The Kestrel solver can be used to solve a GAMS model remotely. For example, consider the transport model. It can be solved locally in GAMS through the following statements,

```plaintext
model transport /all/;
solve transport using lp minimizing z;
```
which specify the \texttt{transport} model and solve it with the default linear programming solver. We can add an option statement to the code to explicitly specify the solver. For example, if we change the linear programming solver to GAMS/MINOS, the code becomes

\begin{verbatim}
model transport /all/;
option lp=minos;
solve transport using lp minimizing z;
\end{verbatim}

To solve the same problem remotely through the NEOS Server, we simply change the linear programming solver to Kestrel. In addition we have to provide an email address. This can be specified either in an option file or by setting the environment variable \texttt{NEOS_EMAIL}. In order to set this permanently, it is recommended to add \texttt{NEOS_EMAIL} to your \texttt{gamsconfig.yaml} file.

Without an option file Kestrel will submit the model instance to the NEOS server and use the default LP solver on NEOS.

\begin{verbatim}
option lp=kestrel;
solve transport using lp minimizing z;
\end{verbatim}

We can support a Kestrel option file and determine the solver on the NEOS server:

\begin{verbatim}
transport.optfile=1;
option lp=kestrel;
solve transport using lp minimizing z;
$echo email jdoe@jeangreyhigh.edu > kestrel.opt
$echo kestrel_solver minos >> kestrel.opt
\end{verbatim}

The statement \texttt{transport.optfile=1} specifies that an options file, called \texttt{kestrel.opt}, will be used. The options file contains the email address, the remote solver name as well as any options for the remote solver. We instruct the Kestrel solver to use GAMS/MINOS as the remote solver by writing the following \texttt{kestrel.opt} file:

\begin{verbatim}
email jdoe@jeangreyhigh.edu
kestrel_solver minos
\end{verbatim}

If you do not know what solvers are available via GAMS/KESTREL on NEOS, submitting a job with a nonexistent solver set will return a list of enabled solvers. If you want to change the URL of the server, you can specify the option \texttt{neos_server}. The complete format of the parameter is \texttt{protocol://host:port}.

A subsequent run of the code through the GAMS interpreter results in the transport model being solved through the NEOS Server with the GAMS/MINOS solver. Once the job is submitted to the NEOS Server, a job number, password, and Web address are displayed to the screen, which provide information on accessing the job and viewing the intermediate output, for example,

--- Executing KESTREL
Job has been submitted to Kestrel
Kestrel/NEOS Job number : 93478
Kestrel/NEOS Job password : utiwtxTK
Check the following URL for progress report :
https://neos-server.org/neos/cgi-bin/nph-neos-solver.cgi?admin=results&jobnumber=93478&pass=utiwtxTK
To look at the solver's output while it is running, point your browser at the URL given in the Kestrel output as shown above, and click on View Intermediate Results in the web page that appears. This will take you to another page that shows all of the output produced by the solver for your problem so far. To track the solver’s progress, simply update this page periodically.

If the NEOS Server or the network becomes unavailable after the submission, a particular job can be retrieved by setting both the kestrel job and kestrel password in the options file.

```plaintext
kestrel_solver minos
kestrel_job 93478
kestrel_password utiwtxTK
```

Re-issuing the command `gams transport` with this options file will retrieve the results for the specified job number.

By specifying `neos_username` and `neos_user_password` in the option file, you can submit an authenticated job using your NEOS user account. Authenticated jobs will appear in your user account on the NEOS website.

```plaintext
neos_username username
neos_user_password password
```

### 5.22.2.1 Using GAMS/KESTREL with IBM DOcloud

In order to use GAMS/Kestrel to solve model instances with Cplex on IBM's DOcloud, the only thing that needs to be done is to provide a different option file. The option `docloud_url` triggers the submission to IBM's DOcloud rather than NEOS. You need an account on DOcloud (and pay IBM for the Cplex solution time in their cloud). Some additional account information needs to be specified in the Kestrel option `docloud_key`. The use of Cplex in DOcloud is limited to solve a model instance, special options that are available in the GAMS/Cplex system, like IIS or FeasOpt cannot be used (yet) through Kestrel/DOcloud. Moreover, for problems with discrete variables, only the primal solution is returned. If you want to specify Cplex options this needs to be done with a Cplex parameter file which is quite different from the GAMS/Cplex option file. You can use a GAMS/Cplex option file and have the additional option `WriteParam` to produce such a Cplex parameter file. The Cplex parameter file might have a few more options compared to the GAMS/Cplex option file because some GAMS options set Cplex parameters. Please note that you can produce the Cplex parameter file with GAMS/Cplex even if you do not have a license for GAMS/Cplex as long as you run a demo sized model (e.g. from the GAMS Model Library) to produce the Cplex parameter file. The following GAMS/Cplex optfile

```plaintext
lpmethod 4
advind 0
eprhs 1e-4
scaind 1
writeparam cplex.prm
```

produces the Cplex parameter file `cplex.prm` which can be submitted to Kestrel/DOcloud via the option `docloud_prmfile`:

```plaintext
CPLEX Parameter File Version 12.6.3.0
CPX_PARAM_ADVIND 0
CPX_PARAM_EPRHS 1.00000000000000e-04
CPX_PARAM_SIMDISPLAY 2
CPX_PARAM_SCAIND 1
CPX_PARAM_TILIM 1.00000000000000e+03
CPX_PARAM_LPMETHOD 4
CPX_PARAM_THREADS 1
CPX_PARAM_PARALLELMODE 1
```
Users of GAMS/Kestrel on platforms other than Windows, Linux, or macOS have to manually provide a Python 3 installation that has access to the following modules:

- GAMS Python API
- IBM DOcloud Python package.

5.23 KNITRO

5.23.1 Introduction

Artelys Knitro is a software package for finding local solutions of both continuous (i.e. smooth) optimization problems, with or without constraints, and discrete optimization problems with integer or binary variables. Even though Knitro has been designed for solving large-scale general problems, it is efficient for solving all of the following classes of optimization problems:

- unconstrained,
- bound constrained,
- equality constrained,
- systems of nonlinear equations,
- least squares problems,
- linear programming problems (LPs),
- quadratic programming problems (QP),
- general (inequality) constrained problems,
- (convex) mixed integer nonlinear programs (MINLP) of moderate size.

The Knitro package provides the following features:

- Efficient and robust solution of small or large problems,
- Solvers for both continuous and discrete problems,
- Derivative-free, 1st derivative and 2nd derivative options,
- Both interior-point (barrier) and active-set optimizers,
- Both feasible and infeasible versions,
- Both iterative and direct approaches for computing steps,

The problems solved by Knitro have the form

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad c^L \leq c(x) \leq c^U \\
& \quad b^L \leq x \leq b^U,
\end{align*}
\]

(1a) (1b) (1c)

where the variables $x$ can be continuous, binary, or integer. This allows many forms of constraints, including bounds on the variables. Knitro requires that the functions $f(x)$ and $c(x)$ be smooth functions.
Knitro implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimization problems. In the interior method (also known as a barrier method), the nonlinear programming problem is replaced by a series of barrier sub-problems controlled by a barrier parameter $\mu$. The algorithm uses trust regions and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter, and repeats the process until the original problem (1) has been solved to the desired accuracy.

Knitro provides two procedures for computing the steps within the interior point approach. In the version known as Interior/CG each step is computed using a projected conjugate gradient iteration. This approach differs from most interior methods proposed in the literature in that it does not compute each step by solving a linear system involving the KKT (or primal-dual) matrix. Instead, it factors a projection matrix, and uses the conjugate gradient method, to approximately minimize a quadratic model of the barrier problem.

The second procedure for computing the steps, which we call Interior/Direct, always attempts to compute a new iterate by solving the primal-dual KKT matrix using direct linear algebra. In the case when this step cannot be guaranteed to be of good quality, or if negative curvature is detected, then the new iterate is computed by the Interior/CG procedure.

Knitro also implements an active-set sequential linear-quadratic programming (SLQP) algorithm which we call Active. This method is similar in nature to a sequential quadratic programming method but uses linear programming sub-problems to estimate the active-set at each iteration. This active-set code may be preferable when a good initial point can be provided, for example, when solving a sequence of related problems.

For problems with discrete variables, Knitro provides two variants of the branch and bound algorithm. The first is a standard implementation, while the second is specialized for convex, mixed-integer nonlinear problems.

We encourage the user to try all algorithmic options to determine which one is more suitable for the application at hand. For guidance on choosing the best algorithm see section Algorithm Options.

For a detailed description of the algorithm implemented in Interior/CG see [30] and for the global convergence theory see [31]. The method implemented in Interior/Direct is described in [198]. The Active algorithm is described in [33] and the global convergence theory for this algorithm is in [34]. An important component of Knitro is the HSL routine MA27 [92] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the Active Set algorithm in Knitro may make use of the COIN-OR Clp linear programming solver module. The version used in Knitro may be downloaded from http://www.artelys.com/tools/clp/.

5.23.2 Usage

Basic details of solver usage, including how to choose Knitro as the solver and how to use a solver-specific option file, are part of Chapter Solver Usage.

As an NLP solver, Knitro can also be used to solve linear programs (LP), and both convex and nonconvex quadratic programs (QCP).
5.23.3 GAMS Options

The following GAMS options are used by the GAMS/Knitro link:

- **Option ResLim = x;**
  
  Sets the time limit in seconds. If this limit is exceeded the solver will terminate and pass on the current solution to GAMS. See also reslim in section GAMS options.

- **Option SysOut = On;**
  
  This option sends additional Knitro messages to the GAMS listing file. It is useful in case of a solver failure or to get algorithmic details. See also sysout in section GAMS options.

- **ModelName.optCA = x;**
  
  Absolute gap stop criterion for a discrete problem. The Knitro option mip_opt_gap_abs takes its default from this value. See also optca in section GAMS options.

- **ModelName.optCR = x;**
  
  Relative gap stop criterion for a discrete problem. The Knitro option mip_opt_gap_rel takes its default from this value. See also optcr in section GAMS options.

5.23.4 Summary of Knitro Options

The Knitro options file *knitro.opt* allows the user to easily set options controlling Knitro's behavior. Options are set by specifying a keyword and a corresponding value on a line in the *knitro.opt* file. Lines that begin with a # character are treated as comments and blank lines are ignored. For example, to set the maximum allowable number of iterations to 500, one could use the following options file:

```
mxit 500
```

5.23.4.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>Indicates which algorithm to use to solve the problem</td>
<td>0</td>
</tr>
<tr>
<td>blasoption</td>
<td>Specifies the BLAS/LAPACK function library to use for basic vector and matrix computations</td>
<td>1</td>
</tr>
<tr>
<td>blasoptionlib</td>
<td>Specifies a dynamic library name that contains object code for BLAS/LAPACK functions</td>
<td></td>
</tr>
<tr>
<td>bndrange</td>
<td>Specifies max limits on the magnitude of constraint and variable bounds</td>
<td>1e+20</td>
</tr>
<tr>
<td>cg_maxit</td>
<td>Determines the maximum allowable number of inner conjugate gradient (CG) iterations per Knitro minor iteration</td>
<td>-1</td>
</tr>
<tr>
<td>cg_pmem</td>
<td>Specifies the amount of nonzero elements per column of the Hessian of the Lagrangian which are retained when computing the incomplete Cholesky preconditioner</td>
<td>10</td>
</tr>
<tr>
<td>cg_precond</td>
<td>Specifies whether an incomplete Cholesky preconditioner is applied during CG iterations in barrier algorithms</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>cg_stoptol</td>
<td>Specifies the relative stopping tolerance used for the conjugate gradient (CG) subproblem solves</td>
<td>0.01</td>
</tr>
<tr>
<td>convex</td>
<td>Declare the problem as convex by setting to 1 or non-convex by setting to 0</td>
<td>-1</td>
</tr>
<tr>
<td>cpuplatform</td>
<td>This option can be used to specify the target instruction set architecture for the machine on which Knitro is running</td>
<td>-1</td>
</tr>
<tr>
<td>datacheck</td>
<td>Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure)</td>
<td>0</td>
</tr>
<tr>
<td>delta</td>
<td>Specifies the initial trust region radius scaling factor used to determine the initial trust region size</td>
<td>1</td>
</tr>
<tr>
<td>honorbnds</td>
<td>Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization</td>
<td>-1</td>
</tr>
<tr>
<td>initpenalty</td>
<td>Specifies the initial penalty parameter used in the Knitro merit functions</td>
<td>10</td>
</tr>
<tr>
<td>initvalues</td>
<td>Enable use of initial guess for levels and marginals (subsequent solves)</td>
<td>1</td>
</tr>
<tr>
<td>initvalues0</td>
<td>Enable use of initial guess for levels and marginals (first solve)</td>
<td>2</td>
</tr>
<tr>
<td>linesearch</td>
<td>Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate</td>
<td>0</td>
</tr>
<tr>
<td>linesearch_maxtrials</td>
<td>Indicates the maximum allowable number of trial points during the linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step</td>
<td>3</td>
</tr>
<tr>
<td>linsolver</td>
<td>Indicates which linear solver to use to solve linear systems arising in Knitro algorithms</td>
<td>0</td>
</tr>
<tr>
<td>linsolver_maxitref</td>
<td>Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro</td>
<td>2</td>
</tr>
<tr>
<td>linsolver_ooc</td>
<td>Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when linsolver = mklpardiso</td>
<td>0</td>
</tr>
<tr>
<td>linsolver_pivottol</td>
<td>Specifies the initial pivot threshold used in factorization routines</td>
<td>1e-08</td>
</tr>
<tr>
<td>names</td>
<td>Enable to pass variable and equation names to Knitro</td>
<td>0</td>
</tr>
<tr>
<td>ncvx_qcqp_init</td>
<td>Specifies the initialization strategy used for non-convex QPs and QCQPs</td>
<td>-1</td>
</tr>
<tr>
<td>newpoint</td>
<td>Specifies additional action to take after every iteration in a solve of a continuous problem</td>
<td>0</td>
</tr>
<tr>
<td>objrange</td>
<td>Specifies the extreme limits of the objective function for purposes of determining unboundedness</td>
<td>1e+20</td>
</tr>
<tr>
<td>option_file</td>
<td>additional option file name - read only by KNITRO solver lib</td>
<td></td>
</tr>
<tr>
<td>output_time</td>
<td>print output on where time is used</td>
<td>0</td>
</tr>
<tr>
<td>presolve</td>
<td>Determine whether or not to use the Knitro presolver to try to simplify the model by removing variables or constraints</td>
<td>1</td>
</tr>
<tr>
<td>presolveop_tighten</td>
<td>Determine whether or not to enable the Knitro presolve operation to tighten variable bounds</td>
<td>-1</td>
</tr>
<tr>
<td>presolve_initpt</td>
<td>Control whether the Knitro presolver can shift a user-supplied initial point</td>
<td>-1</td>
</tr>
<tr>
<td>presolve_level</td>
<td>Set the level of presolve operations to enable through the Knitro presolver</td>
<td>-1</td>
</tr>
<tr>
<td>presolve_passes</td>
<td>Set a maximum limit on the number of passes through the Knitro presolve operations</td>
<td>10</td>
</tr>
</tbody>
</table>
### 5.23.4.2 Derivative options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfgs_scaling</td>
<td>Specify the initial scaling to use for the BFGS or L-BFGS Hessian approximation</td>
<td>0</td>
</tr>
<tr>
<td>findiff_estnoise</td>
<td>This option can be used to enable an estimate of the noise in the model when using finite-difference gradients</td>
<td>0</td>
</tr>
<tr>
<td>findiff_relstepsize</td>
<td>Specifies the relative stepsize used for finite-difference gradients during the optimization</td>
<td>0</td>
</tr>
<tr>
<td>findiff_terminate</td>
<td>This option specifies the termination criteria when using finite-difference gradients</td>
<td>1</td>
</tr>
<tr>
<td>gradopt</td>
<td>Specifies how to compute the gradients of the objective and constraint functions</td>
<td>1</td>
</tr>
<tr>
<td>hessopt</td>
<td>Specifies how to compute the (approximate) Hessian of the Lagrangian</td>
<td>0</td>
</tr>
<tr>
<td>lmsize</td>
<td>Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.23.4.3 Termination options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>feastol</td>
<td>Specifies the final relative stopping tolerance for the feasibility error</td>
<td>1e-06</td>
</tr>
<tr>
<td>feastolabs</td>
<td>Specifies the final absolute stopping tolerance for the feasibility error</td>
<td>0.001</td>
</tr>
<tr>
<td>fstopval</td>
<td>Used to implement a custom stopping condition based on the objective function value</td>
<td>maxdouble</td>
</tr>
<tr>
<td>ftol</td>
<td>The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations</td>
<td>1e-15</td>
</tr>
<tr>
<td>ftol_iters</td>
<td>The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations</td>
<td>5</td>
</tr>
<tr>
<td>infeasol</td>
<td>Specifies the (relative) tolerance used for declaring infeasibility of a model</td>
<td>1e-08</td>
</tr>
<tr>
<td>infeasol_iters</td>
<td>The optimization process will terminate if the relative change in the feasibility error is less than infeasol for infeasol_iters consecutive infeasible iterations</td>
<td>50</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td><code>maxfevals</code></td>
<td>Specifies the maximum number of function evaluations before termination</td>
<td>-1</td>
</tr>
<tr>
<td><code>maxit</code></td>
<td>Specifies the maximum number of iterations before termination</td>
<td>GAMS <code>iterlim</code></td>
</tr>
<tr>
<td><code>maxtime_cpu</code></td>
<td>Specifies, in seconds, the maximum allowable CPU time before termination</td>
<td>1000000000</td>
</tr>
<tr>
<td><code>maxtime_real</code></td>
<td>Specifies, in seconds, the maximum allowable real time before termination</td>
<td>GAMS <code>reslim</code></td>
</tr>
<tr>
<td><code>opttol</code></td>
<td>Specifies the final relative stopping tolerance for the KKT (optimality) error</td>
<td><code>1e-06</code></td>
</tr>
<tr>
<td><code>opttolabs</code></td>
<td>Specifies the final absolute stopping tolerance for the KKT (optimality) error</td>
<td><code>0.001</code></td>
</tr>
<tr>
<td><code>xtol</code></td>
<td>The optimization process will terminate if the relative change in all components of the solution point estimate is less than <code>xtol</code> for <code>xtol_iters</code></td>
<td><code>1e-12</code></td>
</tr>
<tr>
<td><code>xtol_iters</code></td>
<td>The optimization process will terminate if the relative change in the solution estimate is less than <code>xtol</code> for <code>xtol_iters</code> consecutive iterations</td>
<td><code>0</code></td>
</tr>
</tbody>
</table>

### 5.23.4.4 Barrier options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bar_conic_enable</code></td>
<td>Enable special treatments for conic constraints when using the Interior/Direct algorithm (has no affect when using the Interior/CG algorithm)</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_directinterval</code></td>
<td>Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_feasible</code></td>
<td>Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms</td>
<td>0</td>
</tr>
<tr>
<td><code>bar_feasmodetol</code></td>
<td>Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible</td>
<td><code>0.0001</code></td>
</tr>
<tr>
<td><code>bar_initmu</code></td>
<td>Specifies the initial value for the barrier parameter $\mu$ used with the barrier algorithms</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_initpi_mpec</code></td>
<td>Specifies the initial value for the MPEC penalty parameter $\pi$ used when solving problems with complementarity constraints using the barrier algorithms</td>
<td>0</td>
</tr>
<tr>
<td><code>bar_initpt</code></td>
<td>Indicates initial point strategy for $x$, slacks and multipliers when using a barrier algorithm</td>
<td>0</td>
</tr>
<tr>
<td><code>bar_linsys</code></td>
<td>Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_linsys_storage</code></td>
<td>Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_maxcorrectors</code></td>
<td>Specifies the maximum number of corrector steps allowed for primal-dual steps</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_maxcrossit</code></td>
<td>Specifies the maximum number of crossover iterations before termination</td>
<td>0</td>
</tr>
<tr>
<td><code>bar_maxrefactor</code></td>
<td>Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step</td>
<td>-1</td>
</tr>
<tr>
<td><code>bar_mpec_heuristic</code></td>
<td>Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm</td>
<td>0</td>
</tr>
<tr>
<td><code>bar_murule</code></td>
<td>Indicates which strategy to use for modifying the barrier parameter $\mu$ in the barrier algorithms</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>bar_penaltycons</td>
<td>Indicates whether a penalty approach is applied to the constraints</td>
<td>-1</td>
</tr>
<tr>
<td>bar_penaltyrule</td>
<td>Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate</td>
<td>0</td>
</tr>
<tr>
<td>bar_refinement</td>
<td>Specifies whether to try to refine the barrier solution for better precision</td>
<td>0</td>
</tr>
<tr>
<td>bar_relaxcons</td>
<td>Indicates whether a relaxation approach is applied to the constraints</td>
<td>2</td>
</tr>
<tr>
<td>bar_slackboundpush</td>
<td>Specifies the amount by which the barrier slack variables are initially pushed inside the bounds</td>
<td>-1</td>
</tr>
<tr>
<td>bar_switchobj</td>
<td>Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase</td>
<td>1</td>
</tr>
<tr>
<td>bar_switchrule</td>
<td>Indicates whether or not the barrier algorithms will allow switching from an optimality phase to a pure feasibility phase</td>
<td>-1</td>
</tr>
<tr>
<td>bar_watchdog</td>
<td>Specifies whether to enable watchdog heuristic for barrier algorithms</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.23.4.5 Active-set options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>act_lpfeastol</td>
<td>Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms</td>
<td>1e-08</td>
</tr>
<tr>
<td>act_lppenalty</td>
<td>Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms</td>
<td>1</td>
</tr>
<tr>
<td>act_lppresolve</td>
<td>Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms</td>
<td>0</td>
</tr>
<tr>
<td>act_parametric</td>
<td>Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms</td>
<td>1</td>
</tr>
<tr>
<td>act_qpalg</td>
<td>Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms</td>
<td>0</td>
</tr>
<tr>
<td>act_qppenalty</td>
<td>Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.23.4.6 MIP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mip_branchrule</td>
<td>Specifies which branching rule to use for MIP branch and bound procedure</td>
<td>0</td>
</tr>
<tr>
<td>mip_clique</td>
<td>Specifies rules for adding clique cuts</td>
<td>0</td>
</tr>
<tr>
<td>mip_cutfactor</td>
<td>This value specifies a limit on the number of cuts added to a node subproblem</td>
<td>1</td>
</tr>
<tr>
<td>mip_cutoff</td>
<td>This value specifies the objective cutoff value for MIP</td>
<td>maxdouble</td>
</tr>
<tr>
<td>mip_cutting_plane</td>
<td>Specifies when to apply the cutting plane procedure</td>
<td>0</td>
</tr>
<tr>
<td>mip_gub_branch</td>
<td>Specifies whether or not to branch on generalized upper bounds (GUBs)</td>
<td>0</td>
</tr>
<tr>
<td>mip_heuristic_diving</td>
<td>Specifies whether or not to enable the MIP diving heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>mip_heuristic_feaspump</code></td>
<td>Specifies whether or not to enable the MIP feasibility pump heuristic</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_heuristic_lns</code></td>
<td>Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_heuristic_maxit</code></td>
<td>Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled</td>
<td>100</td>
</tr>
<tr>
<td><code>mip_heuristic_misqp</code></td>
<td>Specifies whether or not to enable the MIP MISQP heuristic</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_heuristic_mpec</code></td>
<td>Specifies whether or not to enable the MIP MPEC heuristic</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_heuristic_strategy</code></td>
<td>Specifies the level of effort applied for the MIP heuristic search used to try to find an initial integer feasible point</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_heuristic_terminate</code></td>
<td>Specifies the condition for terminating the MIP heuristic</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_implications</code></td>
<td>Specifies whether or not to add constraints to the MIP derived from logical implications</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_integer_tol</code></td>
<td>This value specifies the threshold for deciding whether or not a variable is determined to be an integer</td>
<td>1e-08</td>
</tr>
<tr>
<td><code>mip_intvar_strategy</code></td>
<td>Specifies how to handle integer variables</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_knapsack</code></td>
<td>Specifies rules for adding MIP knapsack cuts</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_liftproject</code></td>
<td>Specifies rules for adding lift and project cuts</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_lpalg</code></td>
<td>Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch and bound procedure</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_maxnodes</code></td>
<td>Specifies the maximum number of nodes explored (0 means no limit)</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td><code>mip_maxsolves</code></td>
<td>Specifies the maximum number of subproblem solves allowed (0 means no limit)</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_maxtime_cpu</code></td>
<td>Specifies the maximum allowable CPU time in seconds for the complete MIP solution</td>
<td>100000000</td>
</tr>
<tr>
<td><code>mip_maxtime_real</code></td>
<td>Specifies the maximum allowable real time in seconds for the complete MIP solution</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td><code>mip_method</code></td>
<td>Specifies which MIP method to use</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_mir</code></td>
<td>Specifies rules for adding mixed integer rounding cuts</td>
<td>-1</td>
</tr>
<tr>
<td><code>mip_multistart</code></td>
<td>Use to enable MIP multi-start at the branch-and-bound level</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_nodealg</code></td>
<td>Specifies which algorithm to use for standard node subproblem solves in MIP (same options as <code>algorithm</code> user option)</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_numthreads</code></td>
<td>Specify the number of threads to use for MIP branch and bound (when <code>mip_method</code> = 1)</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_opt_gap_abs</code></td>
<td>The absolute optimality gap stop tolerance for MIP</td>
<td>GAMS optca</td>
</tr>
<tr>
<td><code>mip_opt_gap_rel</code></td>
<td>The relative optimality gap stop tolerance for MIP</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td><code>mip_outinterval</code></td>
<td>Specifies node printing interval for <code>mip_outlevel</code> when <code>mip_outlevel</code> &gt; 0</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_outlevel</code></td>
<td>Specifies how much MIP information to print</td>
<td>2</td>
</tr>
<tr>
<td><code>mip_pseudoinit</code></td>
<td>Specifies the method used to initialize pseudo-costs corresponding to variables that have not yet been branched on in the MIP method</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>mip_relaxable</td>
<td>Specifies whether integer variables are relaxable</td>
<td>1</td>
</tr>
<tr>
<td>mip_restart</td>
<td>Specifies whether to enable the MIP restart procedure</td>
<td>1</td>
</tr>
<tr>
<td>mip_rootalg</td>
<td>Specifies which algorithm to use for the root node solve in MIP</td>
<td>0</td>
</tr>
<tr>
<td>mip_rounding</td>
<td>Specifies the MIP rounding rule to apply</td>
<td>-1</td>
</tr>
<tr>
<td>mip_selectdir</td>
<td>Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch and bound tree</td>
<td>0</td>
</tr>
<tr>
<td>mip_selectrule</td>
<td>Specifies the MIP select rule for choosing the next node in the branch and bound tree</td>
<td>0</td>
</tr>
<tr>
<td>mip_strong_candlim</td>
<td>Specifies the maximum number of candidates to explore for MIP strong branching</td>
<td>128</td>
</tr>
<tr>
<td>mip_strong_level</td>
<td>Specifies the maximum number of tree levels on which to perform MIP strong branching</td>
<td>10</td>
</tr>
<tr>
<td>mip_strong_maxit</td>
<td>Specifies the maximum number of iterations to allow for MIP strong branching solves</td>
<td>1000</td>
</tr>
<tr>
<td>mip_terminate</td>
<td>Specifies conditions for terminating the MIP algorithm</td>
<td>0</td>
</tr>
<tr>
<td>mip_zerohalf</td>
<td>Specifies rules for adding zero-half cuts</td>
<td>0</td>
</tr>
</tbody>
</table>

5.23.4.7 Multi-algorithm options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma_maxtime_cpu</td>
<td>Specifies, in seconds, the maximum allowable CPU time before termination for the multi-algorithm (&quot;MA&quot;) procedure (algorithm=5)</td>
<td>100000000</td>
</tr>
<tr>
<td>ma_maxtime_real</td>
<td>Specifies, in seconds, the maximum allowable real time before termination for the multi-algorithm (&quot;MA&quot;) procedure (algorithm=5)</td>
<td>100000000</td>
</tr>
<tr>
<td>ma_outsub</td>
<td>Enable writing algorithm output to files for the multi-algorithm (algorithm=5) procedure</td>
<td>0</td>
</tr>
<tr>
<td>ma_terminate</td>
<td>Define the termination condition for the multi-algorithm (algorithm=5) procedure</td>
<td>1</td>
</tr>
</tbody>
</table>

5.23.4.8 Multi-start options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms_deterministic</td>
<td>Indicates whether Knitro multi-start procedure will be deterministic (when ms_terminate = 0)</td>
<td>1</td>
</tr>
<tr>
<td>ms_enable</td>
<td>Indicates whether Knitro will solve from multiple start points to find a better local minimum</td>
<td>0</td>
</tr>
<tr>
<td>ms_maxbndrange</td>
<td>Specifies the maximum range that an unbounded variable can take when determining new start points</td>
<td>1000</td>
</tr>
<tr>
<td>ms_maxsolves</td>
<td>Specifies how many start points to try in multi-start</td>
<td>0</td>
</tr>
<tr>
<td>ms_maxtime_cpu</td>
<td>Specifies, in seconds, the maximum allowable CPU time before termination</td>
<td>100000000</td>
</tr>
<tr>
<td>ms_maxtime_real</td>
<td>Specifies, in seconds, the maximum allowable real time before termination</td>
<td>100000000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ms_numthreads</td>
<td>Specify the number of threads to use for multi-start (when ms_enable = 1)</td>
<td>0</td>
</tr>
<tr>
<td>ms_num_to_save</td>
<td>Specifies the number of distinct feasible points to save in a file named knitro_mspoints</td>
<td>0</td>
</tr>
<tr>
<td>ms_outsub</td>
<td>Enable writing algorithm output to files for the parallel multi-start procedure</td>
<td>0</td>
</tr>
<tr>
<td>ms_savetol</td>
<td>Specifies the tolerance for deciding if two feasible points are distinct</td>
<td>1e-06</td>
</tr>
<tr>
<td>ms_seed</td>
<td>Seed value used to generate random initial points in multi-start; should be a non-negative integer</td>
<td>0</td>
</tr>
<tr>
<td>ms_startptrange</td>
<td>Specifies the maximum range that each variable can take when determining new start points</td>
<td>1e+20</td>
</tr>
<tr>
<td>msterminate</td>
<td>Specifies the condition for terminating multi-start</td>
<td>0</td>
</tr>
</tbody>
</table>

5.23.4.9 Parallelism options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>blas_numthreads</td>
<td>Specify the number of threads to use for BLAS operations when blasoption = 1</td>
<td>0</td>
</tr>
<tr>
<td>conic_numthreads</td>
<td>Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1)</td>
<td>0</td>
</tr>
<tr>
<td>linsolver_numthreads</td>
<td>Specify the number of threads to use for linear system solve operations when linsolver = 6</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>default thread count</td>
<td>GAMS threads</td>
</tr>
</tbody>
</table>

5.23.4.10 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outappend</td>
<td>Specifies whether output should be started in a new file, or appended to existing files</td>
<td>0</td>
</tr>
<tr>
<td>oudir</td>
<td>Specifies a single directory as the location to write all output files</td>
<td></td>
</tr>
<tr>
<td>outlev</td>
<td>Controls the level of output produced by Knitro</td>
<td>2</td>
</tr>
<tr>
<td>outmode</td>
<td>Specifies where to direct the output from Knitro</td>
<td>0</td>
</tr>
<tr>
<td>outname</td>
<td>Use to specify a custom filename when output is written to a file using outmode</td>
<td></td>
</tr>
<tr>
<td>out_csvinfo</td>
<td>Controls whether or not to generates a file knitro_solve</td>
<td>0</td>
</tr>
<tr>
<td>out_csvname</td>
<td>Use to specify a custom csv filename when using out_csvinfo</td>
<td></td>
</tr>
<tr>
<td>out_hints</td>
<td>Specifies whether to print diagnostic hints (e.g. about user option settings) after solving</td>
<td>1</td>
</tr>
</tbody>
</table>

5.23.4.11 Tuner options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuner</td>
<td>Indicates whether to invoke the Knitro-Tuner</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tuner_maxtime_cpu</td>
<td>Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner</td>
<td>100000000</td>
</tr>
<tr>
<td>tuner_maxtime_real</td>
<td>Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner</td>
<td>100000000</td>
</tr>
<tr>
<td>tuner_optionsfile</td>
<td>Can be used to specify the location of a Tuner options file</td>
<td></td>
</tr>
<tr>
<td>tuner_outsub</td>
<td>Enable writing additional Tuner subproblem solve output to files for the Knitro-Tuner procedure (tuner=1)</td>
<td>0</td>
</tr>
<tr>
<td>tuner_terminate</td>
<td>Define the termination condition for the Knitro-Tuner procedure (tuner=1)</td>
<td>0</td>
</tr>
</tbody>
</table>

5.23.5 Detailed Descriptions of Knitro Options

**act_lpfeastol (real):** Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms

\[\text{Range: } [0, \infty]\]

Default: 1e-08

**act_lppenalty (integer):** Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(all) penalize all constraints.</td>
</tr>
<tr>
<td>2</td>
<td>(nonlinear) penalize only nonlinear constraints.</td>
</tr>
<tr>
<td>3</td>
<td>(dynamic) dynamically choose which constraints to penalize.</td>
</tr>
</tbody>
</table>

**act_lppresolve (boolean):** Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(off) presolve turned off for LP subproblems.</td>
</tr>
<tr>
<td>1</td>
<td>(on) presolve turned on for LP subproblems.</td>
</tr>
</tbody>
</table>

**act_parametric (integer):** Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms

Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms. A parametric approach will solve a sequence of closely related LPs instead of one LP. It may increase the cost of an active-set iteration, but perhaps lead to convergence in fewer iterations.

Default: 1
**act_qpalg (integer):** Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms

Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms. This option has no effect on the Interior/Direct and Interior/CG algorithms.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) do not use a parametric solve (i.e. solve a single LP).</td>
</tr>
<tr>
<td>1</td>
<td>(maybe) use a parametric solve sometimes.</td>
</tr>
<tr>
<td>2</td>
<td>(yes) always try a parametric solve.</td>
</tr>
</tbody>
</table>

**act_qppenalty (integer):** Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) let Knitro automatically decide.</td>
</tr>
<tr>
<td>0</td>
<td>(none) do not penalize constraints in QP subproblems.</td>
</tr>
<tr>
<td>1</td>
<td>(all) penalize all constraints in QP subproblems.</td>
</tr>
</tbody>
</table>

**algorithm (integer):** Indicates which algorithm to use to solve the problem

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) let Knitro automatically choose an algorithm, based on the problem characteristics.</td>
</tr>
<tr>
<td>1</td>
<td>(direct) use the Interior/Direct algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>(cg) use the Interior/CG algorithm.</td>
</tr>
<tr>
<td>3</td>
<td>(active) use the Active Set algorithm.</td>
</tr>
<tr>
<td>4</td>
<td>(sqp) use the SQP algorithm.</td>
</tr>
<tr>
<td>5</td>
<td>(multi) run all algorithms, perhaps in parallel (see Algorithms).</td>
</tr>
</tbody>
</table>

**bar_conic_enable (integer):** Enable special treatments for conic constraints when using the Interior/Direct algorithm (has no affect when using the Interior/CG algorithm)

Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(none) Do not apply any special treatment for conic constraints.</td>
</tr>
<tr>
<td>1</td>
<td>(soc) Apply special treatments for any Second Order Cone (SOC) constraints identified in the model.</td>
</tr>
</tbody>
</table>

**bar_directinterval** *(integer)*: Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra

Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra. This option is only valid for the Interior/Direct algorithm and may be useful on problems where Knitro appears to be taking lots of conjugate gradient steps. Setting `bar_directinterval` to 0 will try to enforce that only direct steps are taken which may produce better results on some problems.

Range: \{-1, ..., \infty\}

Default: -1

**bar_feasible** *(integer)*: Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms

Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms. Note This option can only be used with the Interior/Direct and Interior/CG algorithms. If `bar_feasible = stay` or `bar_feasible = get_stay`, this will activate the feasible version of Knitro. The feasible version of Knitro will force iterates to strictly satisfy inequalities, but does not require satisfaction of equality constraints at intermediate iterates. This option and the honorbnds option may be useful in applications where functions are undefined outside the region defined by inequalities. The initial point must satisfy inequalities to a sufficient degree; if not, Knitro may generate infeasible iterates and does not switch to the feasible version until a sufficiently feasible point is found. Sufficient satisfaction occurs at a point $x$ if it is true for all inequalities that $cl + tol \leq c(x) \leq cu - tol$ The constant tol is determined by the option `bar_feasmodetol`. If `bar_feasible = get` or `bar_feasible = get_stay`, Knitro will place special emphasis on first trying to get feasible before trying to optimize.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) No special emphasis on feasibility.</td>
</tr>
<tr>
<td>1</td>
<td>(stay) Iterates must satisfy inequality constraints once they become sufficiently feasible.</td>
</tr>
<tr>
<td>2</td>
<td>(get) Special emphasis is placed on getting feasible before trying to optimize.</td>
</tr>
<tr>
<td>3</td>
<td>(get_stay) Implement both options 1 and 2 above.</td>
</tr>
</tbody>
</table>

**bar_feasmodetol** *(real)*: Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible

Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible. The tolerance applies to all inequality constraints in the problem. This option only has an effect if option `bar_feasible = stay` or `bar_feasible = get_stay`.

Range: [0, \infty]

Default: 0.0001
bar_initmu (real): Specifies the initial value for the barrier parameter $\mu$ used with the barrier algorithms

Specifies the initial value for the barrier parameter $\mu$ used with the barrier algorithms. This option has no effect on the Active Set algorithm.

Range: $[-\infty, \infty]$

Default: -1

bar_inipi_mpec (real): Specifies the initial value for the MPEC penalty parameter $\pi$ used when solving problems with complementarity constraints using the barrier algorithms

Specifies the initial value for the MPEC penalty parameter $\pi$ used when solving problems with complementarity constraints using the barrier algorithms. If this value is non-positive, then Knitro uses an internal formula to initialize the MPEC penalty parameter.

Range: $[0, \infty]$

Default: 0

bar_initpt (integer): Indicates initial point strategy for x, slacks and multipliers when using a barrier algorithm

Indicates initial point strategy for x, slacks and multipliers when using a barrier algorithm. Note, this option only alters the initial x values if the user does not specify an initial x. This option has no effect on the Active Set algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(convex) Initialization designed for convex models.</td>
</tr>
<tr>
<td>2</td>
<td>(nearbnd) Initialization strategy that stays closer to the bounds.</td>
</tr>
<tr>
<td>3</td>
<td>(central) Initialization strategy that is more central on double-bounded variables.</td>
</tr>
</tbody>
</table>

bar_linsys (integer): Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps

Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps. Eliminating more elements results in a smaller dimensional linear system (but also one that is, perhaps, less numerically stable). The bounds option may be preferable for very large problems with many bounded variables. The ineq option may generate significant speedups on models where the number of variables is small, but the number of inequality constraints is large.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose the linear system form.</td>
</tr>
<tr>
<td>0</td>
<td>(full) Use the full linear system.</td>
</tr>
<tr>
<td>1</td>
<td>(slacks) Eliminate the slack variables.</td>
</tr>
<tr>
<td>2</td>
<td>(bounds) Eliminate the slack variables and bounds.</td>
</tr>
<tr>
<td>3</td>
<td>(ineqs) Eliminate the slack variables, bounds, and some inequalities.</td>
</tr>
</tbody>
</table>
**bar_linsys_storage** *(integer)*: Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps

Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps. The `lowmem` option uses one storage location for multiple linear systems. As a result it may use much less memory, but also may be less efficient when the Interior/Direct algorithm takes a lot of CG steps. The `normal` option uses separate storage for different linear systems.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose the linear system storage approach.</td>
</tr>
<tr>
<td>1</td>
<td>(lowmem) Use common storage for multiple linear systems.</td>
</tr>
<tr>
<td>2</td>
<td>(normal) Use separate storage for different linear systems.</td>
</tr>
</tbody>
</table>

**bar_maxcorrectors** *(integer)*: Specifies the maximum number of corrector steps allowed for primal-dual steps

Specifies the maximum number of corrector steps allowed for primal-dual steps. If the value is positive and the algorithm used is Interior/Direct, then Knitro may add at most `bar_maxcorrectors` corrector steps to the primal-dual step to try to stay closer to the central path. This may speed up convergence on some models (although it may make the cost per iteration a little more expensive). If the value is negative, Knitro automatically determines the maximum number of corrector steps to apply.

Range: \{-∞, ..., ∞\}

Default: -1

**bar_maxcrossit** *(integer)*: Specifies the maximum number of crossover iterations before termination

Specifies the maximum number of crossover iterations before termination. If the value is positive and the algorithm in operation is Interior/Direct or Interior/CG, then Knitro will crossover to the Active Set algorithm near the solution. The Active Set algorithm will then perform at most `bar_maxcrossit` iterations to get a more exact solution. If the value is 0, no Active Set crossover occurs and the interior-point solution is the final result. If Active Set crossover is unable to improve the approximate interior-point solution, then Knitro will restore the interior-point solution. In some cases (especially on large-scale problems or difficult degenerate problems) the cost of the crossover procedure may be significant – for this reason, crossover is disabled by default. Enabling crossover generally provides a more accurate solution than Interior/Direct or Interior/CG.

Range: \{0, ..., ∞\}

Default: 0

**bar_maxrefactor** *(integer)*: Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step

Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step. If this value is set to -1, it will use a dynamic strategy. These refactorizations are performed if negative curvature is detected in the model. Rather than reverting to a CG step, the Hessian matrix is modified in an attempt to make the subproblem convex and then the KKT system is refactorized. Increasing this value will make the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct
algorithm is taking a large number of CG steps (as indicated by a positive value for “CGits” in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Range: \{-1, ..., \infty\}

Default: \(-1\)

**bar_mpec_heuristic** (boolean): Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm

Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm. In some cases enabling this heuristic can speedup the convergence to the solution and provide a more precise solution on MPEC models (i.e., models with complementarity constraints).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not enable the heuristic for MPEC models.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Enable the heuristic for MPEC models.</td>
</tr>
</tbody>
</table>

**bar_murule** (integer): Indicates which strategy to use for modifying the barrier parameter \(\mu\) in the barrier algorithms

Indicates which strategy to use for modifying the barrier parameter \(\mu\) in the barrier algorithms. Not all strategies are available for both barrier algorithms, as described below. This option has no effect on the Active Set algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(monotone) Monotonically decrease the barrier parameter. Available for both barrier algorithms.</td>
</tr>
<tr>
<td>2</td>
<td>(adaptive) Use an adaptive rule based on the complementarity gap to determine the value of the barrier parameter. Available for both barrier algorithms.</td>
</tr>
<tr>
<td>3</td>
<td>(probing) Use a probing (affine-scaling) step to dynamically determine the barrier parameter. Available only for the Interior/Direct algorithm.</td>
</tr>
<tr>
<td>4</td>
<td>(dampmpc) Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, with safeguards on the corrector step. Available only for the Interior/Direct algorithm.</td>
</tr>
<tr>
<td>5</td>
<td>(fullmpc) Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, without safeguards on the corrector step. Available only for the Interior/Direct algorithm.</td>
</tr>
<tr>
<td>6</td>
<td>(quality) Minimize a quality function at each iteration to determine the barrier parameter. Available only for the Interior/Direct algorithm.</td>
</tr>
</tbody>
</table>

**bar_penaltycons** (integer): Indicates whether a penalty approach is applied to the constraints

Indicates whether a penalty approach is applied to the constraints. Using a penalty approach may be helpful when the problem has degenerate or difficult constraints. It may also help
to more quickly identify infeasible problems, or achieve feasibility in problems with difficult constraints. This option has no effect on the Active Set algorithm.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(none) No constraints are penalized.</td>
</tr>
<tr>
<td>2</td>
<td>(all) A penalty approach is applied to all general constraints.</td>
</tr>
<tr>
<td>3</td>
<td>(equalities) Apply a penalty approach to equality constraints only.</td>
</tr>
</tbody>
</table>

**bar_penaltyrule (integer):** Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate

Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate. This option has no effect on the Active Set algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(single) Use a single penalty parameter in the merit function to weight feasibility versus optimality.</td>
</tr>
<tr>
<td>2</td>
<td>(flex) Use a more tolerant and flexible step acceptance procedure based on a range of penalty parameter values.</td>
</tr>
</tbody>
</table>

**bar_refinement (boolean):** Specifies whether to try to refine the barrier solution for better precision

Specifies whether to try to refine the barrier solution for better precision. If enabled, once the optimality conditions are satisfied, Knitro will apply an additional refinement/postsolve phase to try to obtain more precision in the barrier solution. The effect is similar to the effect of enabling bar_maxcrossit, but it is usually much more efficient since it does not involve switching to the Active Set algorithm.

Default: 0

**bar_relaxcons (integer):** Indicates whether a relaxation approach is applied to the constraints

Indicates whether a relaxation approach is applied to the constraints. Using a relaxation approach may be helpful when the problem has degenerate or difficult constraints. This option has no effect on the Active Set algorithm.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) No constraints are relaxed.</td>
</tr>
<tr>
<td>1</td>
<td>(eqs) A relaxation approach is applied to general equality constraints.</td>
</tr>
<tr>
<td>2</td>
<td>(ineqs) A relaxation approach is applied to general inequality constraints.</td>
</tr>
<tr>
<td>3</td>
<td>(all) A relaxation approach is applied to all general constraints.</td>
</tr>
</tbody>
</table>

**bar_slackboundpush (real):** Specifies the amount by which the barrier slack variables are initially
pushed inside the bounds ←

Specifies the amount by which the barrier slack variables are initially pushed inside the bounds. A smaller value may be preferable when warm-starting from a point close to the solution.

Range: \([-\infty, \infty]\)

Default: \(-1\)

**bar_switchobj** (integer): Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase ←

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) No (or zero) objective.</td>
</tr>
<tr>
<td>1</td>
<td>(scalarprox) Proximal point objective with scalar weighting.</td>
</tr>
<tr>
<td>2</td>
<td>(diagprox) Proximal point objective with diagonal weighting.</td>
</tr>
</tbody>
</table>

**bar_switchrule** (integer): Indicates whether or not the barrier algorithms will allow switching from an optimality phase to a pure feasibility phase ←

Indicates whether or not the barrier algorithms will allow switching from an optimality phase to a pure feasibility phase. This option has no effect on the Active Set algorithm.

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>(auto) Let Knitro determine the switching procedure.</td>
</tr>
<tr>
<td>0</td>
<td>(never) Never switch to feasibility phase.</td>
</tr>
<tr>
<td>2</td>
<td>(moderate) Allow switches to feasibility phase.</td>
</tr>
<tr>
<td>3</td>
<td>(aggressive) Use a more aggressive switching rule.</td>
</tr>
</tbody>
</table>

**bar_watchdog** (boolean): Specifies whether to enable watchdog heuristic for barrier algorithms ←

Specifies whether to enable watchdog heuristic for barrier algorithms. In general, enabling the watchdog heuristic makes the barrier algorithms more likely to accept trial points. Specifically, the watchdog heuristic may occasionally accept trial points that increase the merit function, provided that subsequent iterates decrease the merit function.

Default: 0

**bfgs_scaling** (integer): Specify the initial scaling to use for the BFGS or L-BFGS Hessian approximation ←

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(dynamic) Dynamically choose which scaling to use.</td>
</tr>
<tr>
<td>1</td>
<td>(invhess) The scaling approximates the scale of the inverse Hessian.</td>
</tr>
<tr>
<td>2</td>
<td>(hess) The scaling approximates the scale of the Hessian.</td>
</tr>
</tbody>
</table>
blasoption (integer): Specifies the BLAS/LAPACK function library to use for basic vector and matrix computations

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(knitro) Use Knitro built-in functions.</td>
</tr>
<tr>
<td>1</td>
<td>(intel) Use Intel Math Kernel Library (MKL) functions on available platforms.</td>
</tr>
<tr>
<td>2</td>
<td>(dynamic) Use the dynamic library specified with option blasoptionlib.</td>
</tr>
<tr>
<td>3</td>
<td>(blis) Use BLIS functions on available platforms (currently not available on Windows OS).</td>
</tr>
</tbody>
</table>

blasoptionlib (string): Specifies a dynamic library name that contains object code for BLAS/LAPACK functions

Specifies a dynamic library name that contains object code for BLAS/LAPACK functions. The library must implement all the functions declared in the file include/blas_lapack.h.

blas_numthreads (integer): Specify the number of threads to use for BLAS operations when blasoption = 1

Range: \( \{0, \ldots, \infty\} \)

Default: 0

bndrange (real): Specifies max limits on the magnitude of constraint and variable bounds

Specifies max limits on the magnitude of constraint and variable bounds. Any constraint or variable bounds whose magnitude is greater than or equal to bndrange will be treated as infinite by Knitro. Using very large, finite bounds is discouraged (and is generally an indication of a poorly scaled model).

Range: \([0, \infty]\)

Default: 1e+20

cg_maxit (integer): Determines the maximum allowable number of inner conjugate gradient (CG) iterations per Knitro minor iteration

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Let Knitro automatically determine a value.</td>
</tr>
<tr>
<td>0</td>
<td>Knitro will set a maximum value based on the problem size.</td>
</tr>
<tr>
<td>n</td>
<td>At most n&gt;0 CG iterations may be performed during one minor iteration of Knitro.</td>
</tr>
</tbody>
</table>

cg_pmem (integer): Specifies the amount of nonzero elements per column of the Hessian of the Lagrangian which are retained when computing the incomplete Cholesky preconditioner

Default: 10
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>At most n&gt;0 nonzero elements per column.</td>
</tr>
</tbody>
</table>

**cg_precond** *(boolean)*: Specifies whether an incomplete Cholesky preconditioner is applied during CG iterations in barrier algorithms

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Not applied.</td>
</tr>
<tr>
<td>1</td>
<td>(chol) Preconditioner is applied.</td>
</tr>
</tbody>
</table>

**cg_stoptol** *(real)*: Specifies the relative stopping tolerance used for the conjugate gradient (CG) subproblem solves

Range: $[0, \infty]$  
Default: 0.01

**conic_numthreads** *(integer)*: Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1)

Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1). 0 Let Knitro choose the number of threads (currently sets conic_numthreads based on numthreads). n>0 Use n threads for the conic solver.

Range: $\{0, ..., \infty\}$  
Default: 0

**convex** *(integer)*: Declare the problem as convex by setting to 1 or non-convex by setting to 0

Declare the problem as convex by setting to 1 or non-convex by setting to 0. Otherwise, Knitro will try to determine this automatically, but may only be able to do so for simple model forms such as QPs or QCQPs. If your model is specified as (or automatically determined to be) convex, this will cause Knitro to apply specializations and tunings that are often beneficial for convex models to speed up the solution. Currently this option is only active for the Interior/Direct algorithm, but may be applied to other algorithms in the future.

Range: $\{-1, ..., 1\}$  
Default: -1

**cpuplatform** *(integer)*: This option can be used to specify the target instruction set architecture for the machine on which Knitro is running

This option can be used to specify the target instruction set architecture for the machine on which Knitro is running. This can be used, for example, to try to produce more consistent Knitro performance across various architectures (at the expense of, perhaps, slower performance on some platforms). This option is currently only used for the Intel Math Kernel Library (MKL) functions used inside Knitro.

Range: $\{-1, ..., 5\}$  
Default: -1
**datacheck (boolean):** Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure).

Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure). The datacheck may have a non-trivial cost for large problems.

Default: 0

**delta (real):** Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

Range: $[1e^{-14}, \infty)$

Default: 1

**feastol (real):** Specifies the final relative stopping tolerance for the feasibility error.

Specifies the final relative stopping tolerance for the feasibility error. Smaller values of feastol result in a higher degree of accuracy in the solution with respect to feasibility.

Range: $[0, \infty)$

Default: $1e^{-06}$

**feastolabs (real):** Specifies the final absolute stopping tolerance for the feasibility error.

Synonym: feastol.abs

Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of feastol.abs result in a higher degree of accuracy in the solution with respect to feasibility.

Range: $[0, \infty)$

Default: 0.001

**findiff_estnoise (integer):** This option can be used to enable an estimate of the noise in the model when using finite-difference gradients.

This option can be used to enable an estimate of the noise in the model when using finite-difference gradients. This noise estimate can then be used to set a finite-difference steplength appropriate for the estimated noise level. This can improve performance on models with noise (e.g. noisy black-box optimization models). The cost of the noise estimation procedure is usually a few extra function evaluations.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not enable any noise estimation procedure for finite-difference gradients.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Enable noise estimation procedure for finite-difference gradients.</td>
</tr>
<tr>
<td>2</td>
<td>(withcurv) Enable noise estimation and curvature factor for finite-difference gradients.</td>
</tr>
</tbody>
</table>

**findiff_relstepsize (real):** Specifies the relative step size used for finite-difference gradients during the optimization.
Specifies the relative stepsize used for finite-difference gradients during the optimization. This option sets the stepsize for all variables. Note that this option has no affect on the finite-difference derivatives computed for the derivative checker (default values are always used here). It is only used for the finite-difference derivatives computed during the optimization.

Range: \([-\infty, \infty]\)

Default: 0

`findiff.terminate` (boolean): This option specifies the termination criteria when using finite-difference gradients

This option specifies the termination criteria when using finite-difference gradients. The optimality (or KKT) conditions for nonlinear optimization depend on gradient values of the nonlinear objective and constraint functions. When using finite-difference gradients (e.g. `gradopt > 1`), there will typically be small errors in the computed gradients that will limit the precision in the solution (and the ability to satisfy the optimality conditions). By default, Knitro will try to estimate these finite-difference gradient errors and terminate when it seems that no more accuracy in the solution is possible. The solution will be treated as optimal as long as it is feasible and the optimality conditions are satisfied either by the optimality tolerances (`opttol` and `opttol.abs`) or the error estimates. On some problems, the error estimates may result in extra function evaluations on some iterations, but will often prevent extra iterations that produce no significant improvement in the solution. This special termination can be disabled by setting `findiff.terminate = 0` (none).

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none)  No special criteria; use the standard stopping conditions.</td>
</tr>
<tr>
<td>1</td>
<td>(errest) Allow termination based on estimates of the finite-difference error (when no more significant progress is likely).</td>
</tr>
</tbody>
</table>

`fstopval` (real): Used to implement a custom stopping condition based on the objective function value

Used to implement a custom stopping condition based on the objective function value. Knitro will stop and declare that a satisfactory solution was found if a feasible objective function value at least as good as the value specified by `fstopval` is achieved. This stopping condition is only active when the absolute value of `fstopval` is less than `objrange`.

Range: \([-\infty, \infty]\)

Default: `maxdouble`

`ftol` (real): The optimization process will terminate if the relative change in the objective function is less than `ftol` for `ftol.liters` consecutive feasible iterations

Range: \([0, \infty]\)

Default: `1e-15`

`ftol.liters` (integer): The optimization process will terminate if the relative change in the objective function is less than `ftol` for `ftol.liters` consecutive feasible iterations

Range: \(\{1, ..., \infty\}\)

Default: 5

`gradopt` (integer): Specifies how to compute the gradients of the objective and constraint functions

Default: 1
### hessopt (integer)

Specifies how to compute the (approximate) Hessian of the Lagrangian

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto)</td>
</tr>
<tr>
<td>1</td>
<td>(exact) User provides a routine for computing the exact Hessian.</td>
</tr>
<tr>
<td>2</td>
<td>(bfgs) Knitro computes a (dense) quasi-Newton BFGS Hessian.</td>
</tr>
<tr>
<td>3</td>
<td>(sr1) Knitro computes a (dense) quasi-Newton SR1 Hessian.</td>
</tr>
<tr>
<td>5</td>
<td>(product) User provides a routine to compute the Hessian-vector products.</td>
</tr>
<tr>
<td>6</td>
<td>(lbfgs) Knitro computes a limited-memory quasi-Newton BFGS Hessian (its size is determined by the option lmsize).</td>
</tr>
<tr>
<td>7</td>
<td>(gauss_newton) Knitro computes a Gauss-Newton approximation of the hessian (available for least-squares only, and default value for least-squares)</td>
</tr>
</tbody>
</table>

Default: 0

### honorbnds (integer)

Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization

Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization. This option and the bar_feasible option may be useful in applications where functions are undefined outside the region defined by inequalities.

Default: -1

### infefastol (real)

Specifies the (relative) tolerance used for declaring infeasibility of a model

Specifies the (relative) tolerance used for declaring infeasibility of a model. Smaller values of infefastol make it more difficult to satisfy the conditions Knitro uses for detecting infeasible models. If you believe Knitro incorrectly declares a model to be infeasible, then you should try a smaller value for infefastol.

Range: $[0, \infty]$.

Default: $1e^{-08}$

### infefastol_iters (integer)

The optimization process will terminate if the relative change in the feasibility error is less than infefastol for infefastol_iters consecutive infeasible iterations

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Knitro automatically determine the best setting.</td>
</tr>
<tr>
<td>0</td>
<td>(no) Knitro does not require that the bounds on the variables be satisfied at intermediate iterates.</td>
</tr>
<tr>
<td>1</td>
<td>(always) Knitro enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables.</td>
</tr>
<tr>
<td>2</td>
<td>(initpt) Knitro enforces that the initial point satisfies the bounds on the variables.</td>
</tr>
</tbody>
</table>
1932 Solver Manuals

Range: \{1, ..., \infty\}

Default: 50

\textbf{initpenalty} (\textit{real}): Specifies the initial penalty parameter used in the Knitro merit functions

Specifies the initial penalty parameter used in the Knitro merit functions. The Knitro merit functions are used to balance improvements in the objective function versus improvements in feasibility. A larger initial penalty value places more weight initially on feasibility in the merit function.

Range: \[0, \infty\]

Default: 10

\textbf{initvalues} (\textit{integer}): Enable use of initial guess for levels and marginals (subsequent solves)

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Using no initial values</td>
</tr>
<tr>
<td>1</td>
<td>Using all initial values</td>
</tr>
<tr>
<td>2</td>
<td>Using only non-default initial values</td>
</tr>
</tbody>
</table>

\textbf{initvalues0} (\textit{integer}): Enable use of initial guess for levels and marginals (first solve)

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Using no initial values</td>
</tr>
<tr>
<td>1</td>
<td>Using all initial values</td>
</tr>
<tr>
<td>2</td>
<td>Using only non-default initial values</td>
</tr>
</tbody>
</table>

\textbf{linesearch} (\textit{integer}): Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate

Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate. This option has no effect on the Interior/CG or Active Set algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(backtrack) Use a simple backtracking scheme.</td>
</tr>
<tr>
<td>2</td>
<td>(interpolate) Use a cubic interpolation scheme.</td>
</tr>
<tr>
<td>3</td>
<td>(weakwolfe) Use a linesearch that satisfies the weak Wolfe conditions (unconstrained only).</td>
</tr>
</tbody>
</table>

\textbf{linesearch_maxtrials} (\textit{integer}): Indicates the maximum allowable number of trial points during the linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step
Indicates the maximum allowable number of trial points during the linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step. This option has no effect on the Interior/CG or Active Set algorithm.

Range: \( \{0, ..., \infty\} \)

Default: 3

**linsolver** (integer): Indicates which linear solver to use to solve linear systems arising in Knitro algorithms

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the linear solver.</td>
</tr>
<tr>
<td>1</td>
<td>(internal) Not currently used; reserved for future use. Same as auto for now.</td>
</tr>
<tr>
<td>2</td>
<td>(hybrid) Use a hybrid approach where the solver chosen depends on the particular linear system which needs to be solved.</td>
</tr>
<tr>
<td>3</td>
<td>(qr) Use a dense QR method. This approach uses LAPACK QR routines. Since it uses a dense method, it is only efficient for small problems. It may often be the most efficient method for small problems with dense Jacobians or Hessian matrices.</td>
</tr>
<tr>
<td>4</td>
<td>(ma27) Use the HSL MA27 sparse symmetric indefinite solver.</td>
</tr>
<tr>
<td>5</td>
<td>(ma57) Use the HSL MA57 sparse symmetric indefinite solver.</td>
</tr>
<tr>
<td>6</td>
<td>(mklpardiso) Use the Intel MKL PARDISO (parallel, deterministic) sparse symmetric indefinite solver.</td>
</tr>
<tr>
<td>7</td>
<td>(ma97) Use the HSL MA97 (parallel, deterministic) sparse symmetric indefinite solver.</td>
</tr>
<tr>
<td>8</td>
<td>(ma86) Use the HSL MA86 (parallel, non-deterministic) sparse symmetric indefinite solver.</td>
</tr>
</tbody>
</table>

**linsolver_maxitref** (integer): Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro

Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro. Iterative refinement steps may be applied when there are significant errors (e.g., large residuals) in the linear system solves. Applying more iterative refinement steps may improve the numerical accuracy of the linear solves at extra cost.

Range: \( \{0, ..., \infty\} \)

Default: 2

**linsolver_numthreads** (integer): Specify the number of threads to use for linear system solve operations when **linsolver** = 6

Range: \( \{0, ..., \infty\} \)

Default: 0

**linsolver_ooc** (integer): Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when **linsolver** = mklpardiso

Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when **linsolver** = mklpardiso. This option is only active when **linsolver** = mklpardiso.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not use Intel MKL PARDISO out-of-core option.</td>
</tr>
<tr>
<td>1</td>
<td>(maybe) Maybe solve out-of-core depending on how much space is needed.</td>
</tr>
<tr>
<td>2</td>
<td>(yes) Solve linear systems out-of-core when using Intel MKL PARDISO.</td>
</tr>
</tbody>
</table>

**linsolver_pivottol (real):** Specifies the initial pivot threshold used in factorization routines

Specifies the initial pivot threshold used in factorization routines. The value should be in the range \([0, \ldots, 0.5]\) with higher values resulting in more pivoting (more stable factorizations). Values less than 0 will be set to 0 and values larger than 0.5 will be set to 0.5. If `linsolver_pivottol` is non-positive, initially no pivoting will be performed. Smaller values may improve the speed of the code but higher values are recommended for more stability (for example, if the problem appears to be very ill-conditioned).

Range: \([0, 0.5]\]

Default: \(1e-08\)

**lmsize (integer):** Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option

Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option. The value must be between 1 and 100 and is only used with `hessopt = 6`. Larger values may give a more accurate, but more expensive, Hessian approximation. Smaller values may give a less accurate, but faster, Hessian approximation. When using the limited memory BFGS approach it is recommended to experiment with different values of this parameter.

Range: \(\{1, \ldots, 100\}\)

Default: 10

**maxfevals (integer):** Specifies the maximum number of function evaluations before termination

Specifies the maximum number of function evaluations before termination. Values less than zero imply no limit.

Range: \(\{-\infty, \ldots, \infty\}\)

Default: \(-1\)

**maxit (integer):** Specifies the maximum number of iterations before termination

Synonym: iterlim

Default: GAMS iterlim

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let Knitro automatically choose a value based on the problem type. Currently Knitro sets this value to 10000 for LPs/NLPs and 3000 for MIP problems.</td>
</tr>
<tr>
<td>n</td>
<td>At most (n&gt;0) iterations may be performed before terminating.</td>
</tr>
</tbody>
</table>

**maxtime_cpu (real):** Specifies, in seconds, the maximum allowable CPU time before termination
Range: $[0, 100000000]$
Default: $100000000$

**maxtime_real** *(real)*: Specifies, in seconds, the maximum allowable real time before termination

Synonym: reslim

Range: $[0, 100000000]$
Default: GAMS reslim

**ma_maxtime_cpu** *(real)*: Specifies, in seconds, the maximum allowable CPU time before termination for the multi-algorithm ("MA") procedure *(algorithm=5)*

Range: $[0, \infty]$
Default: $100000000$

**ma_maxtime_real** *(real)*: Specifies, in seconds, the maximum allowable real time before termination for the multi-algorithm ("MA") procedure *(algorithm=5)*

Range: $[0, \infty]$
Default: $100000000$

**ma_outsub** *(boolean)*: Enable writing algorithm output to files for the multi-algorithm *(algorithm=5)* procedure

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not write detailed algorithm output to files.</td>
</tr>
<tr>
<td>1</td>
<td>Write detailed algorithm output to files named knitro_ma_*_.log.</td>
</tr>
</tbody>
</table>

**ma terminate** *(integer)*: Define the termination condition for the multi-algorithm *(algorithm=5)* procedure

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terminate after all algorithms have completed.</td>
</tr>
<tr>
<td>1</td>
<td>Terminate at first locally optimal solution.</td>
</tr>
<tr>
<td>2</td>
<td>Terminate at first feasible solution estimate.</td>
</tr>
<tr>
<td>3</td>
<td>Terminate at first solution estimate of any type.</td>
</tr>
</tbody>
</table>

**mip_branchrule** *(integer)*: Specifies which branching rule to use for MIP branch and bound procedure

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><em>(auto)</em> Let Knitro automatically choose the branching rule.</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>(most_frac) Use most fractional (most infeasible) branching.</td>
</tr>
<tr>
<td>2</td>
<td>(pseudocost) Use pseudo-cost branching.</td>
</tr>
<tr>
<td>3</td>
<td>(strong) Use strong branching (see options <code>mip_strong_candlim</code>, <code>mip_strong_level</code> and <code>mip_strong_maxit</code> for further control of strong branching procedure).</td>
</tr>
</tbody>
</table>

**mip_clique (integer):** Specifies rules for adding clique cuts

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Do not add clique cuts.</td>
</tr>
<tr>
<td>1</td>
<td>(root) Add cuts derived from equalities at the root node only.</td>
</tr>
<tr>
<td>2</td>
<td>(tree) Add cuts derived from equalities at every node depending on the solution of the relaxation and the cut generation strategy.</td>
</tr>
<tr>
<td>3</td>
<td>(all) Add cuts derived from equalities combining both root and tree cut generations.</td>
</tr>
</tbody>
</table>

**mip_cutfactor (real):** This value specifies a limit on the number of cuts added to a node subproblem

This value specifies a limit on the number of cuts added to a node subproblem. If non-negative, a maximum of `mip_cutfactor` times the number of constraints is possibly appended.

Range: \([0, \infty]\)

Default: 1

**mip_cutoff (real):** This value specifies the objective cutoff value for MIP

Range: \([-\infty, \infty]\)

Default: maxdouble

**mip_cutting_plane (boolean):** Specifies when to apply the cutting plane procedure

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) No cutting plane procedure enabled.</td>
</tr>
<tr>
<td>1</td>
<td>(root) Perform cutting plane procedure at the root node only.</td>
</tr>
</tbody>
</table>

**mip_gub_branch (boolean):** Specifies whether or not to branch on generalized upper bounds (GUBs)

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not branch on GUBs.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Allow branching on GUBs.</td>
</tr>
</tbody>
</table>

**mip_heuristic_diving (integer):** Specifies whether or not to enable the MIP diving heuristic


Specifies whether or not to enable the MIP diving heuristic. This option is a bit-valued option where various diving heuristics can be enabled by activating the corresponding bit value as described below. Setting this option to -1 will use an automatic setting and setting the value to 0 will disable all diving heuristics. Otherwise, set this parameter value to the sum of the values for the individual diving heuristics you wish to enable. For example, to enable only the "fractional" and "linesearch" diving heuristics, you would set this option value to 9 (summing 1 for fractional and 8 for linesearch).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro determine automatically from mip_heuristic_strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(bit 0) Enable fractional diving heuristic.</td>
</tr>
<tr>
<td>2</td>
<td>(bit 1) Enable vector length diving heuristic.</td>
</tr>
<tr>
<td>4</td>
<td>(bit 2) Enable coefficient diving heuristic.</td>
</tr>
<tr>
<td>8</td>
<td>(bit 3) Enable linesearch diving heuristic.</td>
</tr>
<tr>
<td>16</td>
<td>(bit 4) Enable guided diving heuristic.</td>
</tr>
</tbody>
</table>

**mip_heuristic_feaspump (integer):** Specifies whether or not to enable the MIP feasibility pump heuristic

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro determine automatically from mip_heuristic_strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(off) Feasibility pump heuristic is not applied.</td>
</tr>
<tr>
<td>1</td>
<td>(on) Feasibility pump heuristic is enabled.</td>
</tr>
</tbody>
</table>

**mip_heuristic_lns (integer):** Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics

Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics. This option is a bit-valued option where various LNS heuristics can be enabled by activating the corresponding bit value as described below. Setting this option to -1 will use an automatic setting and setting the value to 0 will disable all LNS heuristics. Otherwise, set this parameter value to the sum of the values for the individual LNS heuristics you wish to enable. For example, to enable both the "RENS" and "RINS" LNS heuristics, you would set this option value to 3 (summing 1 for RENS and 2 for RINS).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro determine automatically from mip_heuristic_strategy.</td>
</tr>
<tr>
<td>1</td>
<td>(bit 0) Enable relaxation enforced neighborhood search (RENS) heuristic.</td>
</tr>
<tr>
<td>2</td>
<td>(bit 1) Enable relaxation induced neighborhood search (RINS) heuristic.</td>
</tr>
</tbody>
</table>

**mip_heuristic_maxit (integer):** Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled

Range: \{0, ..., \infty\}

Default: 100
mip_heuristic_misqp (integer): Specifies whether or not to enable the MIP MISQP heuristic

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro determine automatically from mip_heuristic_strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(off) MISQP heuristic is not applied.</td>
</tr>
<tr>
<td>1</td>
<td>(on) MISQP heuristic is enabled.</td>
</tr>
</tbody>
</table>

mip_heuristic_mpec (integer): Specifies whether or not to enable the MIP MPEC heuristic

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro determine automatically from mip_heuristic_strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(off) MPEC heuristic is not applied.</td>
</tr>
<tr>
<td>1</td>
<td>(on) MPEC heuristic is enabled.</td>
</tr>
</tbody>
</table>

mip_heuristic_strategy (integer): Specifies the level of effort applied for the MIP heuristic search used to try to find an initial integer feasible point

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro choose the heuristic strategy to apply (if any).</td>
</tr>
<tr>
<td>0</td>
<td>(none) No heuristic search applied.</td>
</tr>
<tr>
<td>1</td>
<td>(basic) Apply basic heuristics.</td>
</tr>
<tr>
<td>2</td>
<td>(advanced) Apply more advanced heuristics.</td>
</tr>
<tr>
<td>3</td>
<td>(extensive) Apply most extensive heuristics.</td>
</tr>
</tbody>
</table>

mip_heuristicTerminate (integer): Specifies the condition for terminating the MIP heuristic

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(feasible) Terminate at first feasible point or iteration limit (whichever comes first).</td>
</tr>
<tr>
<td>2</td>
<td>(limit) Always run to the iteration limit.</td>
</tr>
</tbody>
</table>

mip_implications (boolean): Specifies whether or not to add constraints to the MIP derived from logical implications

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not add constraints from logical implications.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Knitro adds constraints from logical implications.</td>
</tr>
</tbody>
</table>
**mip_integer_tol** *(real)*: This value specifies the threshold for deciding whether or not a variable is determined to be an integer

Range: [0, 1]

Default: 1e-08

**mip_intvar_strategy** *(integer)*: Specifies how to handle integer variables

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) No special treatment applied.</td>
</tr>
<tr>
<td>1</td>
<td>(relax) Relax all integer variables.</td>
</tr>
<tr>
<td>2</td>
<td>(mpec) Convert all binary variables to complementarity constraints.</td>
</tr>
</tbody>
</table>

**mip_knapsack** *(integer)*: Specifies rules for adding MIP knapsack cuts

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Do not add knapsack cuts.</td>
</tr>
<tr>
<td>1</td>
<td>(ineqs) Add knapsack cuts derived from linear (in)equalities at the root node only.</td>
</tr>
<tr>
<td>2</td>
<td>(lifted) Add lifted separation cuts derived at every node from the solution of the relaxation.</td>
</tr>
<tr>
<td>3</td>
<td>(all) Add both root node cuts from (in)equalities and lifted separation cuts.</td>
</tr>
</tbody>
</table>

**mip_liftproject** *(boolean)*: Specifies rules for adding lift and project cuts

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Do not add lift and project cuts.</td>
</tr>
<tr>
<td>1</td>
<td>(root) Add lift and project cuts at the root node only.</td>
</tr>
</tbody>
</table>

**mip_lpalg** *(integer)*: Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch and bound procedure

Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch and bound procedure. LP subproblems may arise if the problem is a mixed integer linear program (MILP), or if using **mip_method** = HQG. (Nonlinear programming subproblems use the algorithm specified by the **algorithm** option.)

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose an algorithm, based on the problem characteristics.</td>
</tr>
<tr>
<td>1</td>
<td>(direct) Use the Interior/Direct (barrier) algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>(cg) Use the Interior/CG (barrier) algorithm.</td>
</tr>
<tr>
<td>3</td>
<td>(active) Use the Active Set (simplex) algorithm.</td>
</tr>
</tbody>
</table>
**mip_maxnodes** *(integer)*: Specifies the maximum number of nodes explored (0 means no limit) →

Synonym: nodlim

Range: \(\{0, \ldots, \infty\}\)

Default: GAMS nodlim

**mip_maxsolves** *(integer)*: Specifies the maximum number of subproblem solves allowed (0 means no limit) →

Range: \(\{0, \ldots, \infty\}\)

Default: 0

**mip_maxtime_cpu** *(real)*: Specifies the maximum allowable CPU time in seconds for the complete MIP solution →

Specifies the maximum allowable CPU time in seconds for the complete MIP solution. Use `maxtime_cpu` to additionally limit time spent per subproblem solve.

Range: \([0, \infty]\)

Default: 100000000

**mip_maxtime_real** *(real)*: Specifies the maximum allowable real time in seconds for the complete MIP solution →

Specifies the maximum allowable real time in seconds for the complete MIP solution. Use `maxtime_real` to additionally limit time spent per subproblem solve.

Range: \([0, \infty]\)

Default: GAMS reslim

**mip_method** *(integer)*: Specifies which MIP method to use →

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro automatically choose the method.</td>
</tr>
<tr>
<td>1</td>
<td>(BB) Use the standard branch and bound method.</td>
</tr>
<tr>
<td>2</td>
<td>(HQG) Use the hybrid Quesada-Grossman method (for convex, nonlinear problems only).</td>
</tr>
<tr>
<td>3</td>
<td>(MISQP) Use mixed-integer SQP method (allows for non-relaxable integer variables).</td>
</tr>
</tbody>
</table>

**mip_mir** *(integer)*: Specifies rules for adding mixed integer rounding cuts →

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro decide whether to add mixed integer rounding cuts.</td>
</tr>
<tr>
<td>0</td>
<td>(none) Do not add mixed integer rounding cuts.</td>
</tr>
</tbody>
</table>
5.23 KNITRO

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(all) Add cuts derived from equalities at every node depending on the solution of the relaxation and the cut generation strategy.</td>
</tr>
<tr>
<td>2</td>
<td>(nlp) Add mixed-integer rounding cuts at the root node and for nonlinear constraints.</td>
</tr>
</tbody>
</table>

**mip_multistart** *(boolean):* Use to enable MIP multi-start at the branch-and-bound level

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(off) Do not enable MIP multi-start for branch-and-bound.</td>
</tr>
<tr>
<td>1</td>
<td>(on) Enable MIP multi-start for branch-and-bound.</td>
</tr>
</tbody>
</table>

**mip_nodealg** *(integer):* Specifies which algorithm to use for standard node subproblem solves in MIP (same options as algorithm user option)

Range: {0, ..., 5}

Default: 0

**mip_numthreads** *(integer):* Specify the number of threads to use for MIP branch and bound (when **mip_method** = 1)

Specify the number of threads to use for MIP branch and bound (when **mip_method** = 1). 0 Let Knitro choose the number of threads. n > 0 Use n threads for the MIP branch and bound.

Range: {0, ..., ∞}

Default: 0

**mip_opt_gap_abs** *(real):* The absolute optimality gap stop tolerance for MIP

Synonym: optca

Range: [-1, ∞]

Default: GAMS optca

**mip_opt_gap_rel** *(real):* The relative optimality gap stop tolerance for MIP

Synonym: optcr

Range: [-1, ∞]

Default: GAMS optcr

**mip_outinterval** *(integer):* Specifies node printing interval for **mip_outlevel** when **mip_outlevel** > 0

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let Knitro decide.</td>
</tr>
<tr>
<td>1</td>
<td>Print output every node.</td>
</tr>
<tr>
<td>2</td>
<td>Print output every 2nd node.</td>
</tr>
<tr>
<td>n</td>
<td>Print output every Nth node.</td>
</tr>
</tbody>
</table>
mip_outlevel (integer): Specifies how much MIP information to print

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Do not print any MIP node information.</td>
</tr>
<tr>
<td>1</td>
<td>(iters) Print one line of output for every node.</td>
</tr>
<tr>
<td>2</td>
<td>(iterstime) Also print accumulated time for every node.</td>
</tr>
<tr>
<td>3</td>
<td>(root) Also print detailed log from root node solve.</td>
</tr>
</tbody>
</table>

mip_pseudoinit (integer): Specifies the method used to initialize pseudo-costs corresponding to variables that have not yet been branched on in the MIP method

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let Knitro automatically choose the method.</td>
</tr>
<tr>
<td>1</td>
<td>Initialize using the average value of computed pseudo-costs.</td>
</tr>
<tr>
<td>2</td>
<td>Initialize using strong branching.</td>
</tr>
</tbody>
</table>

mip_relaxable (boolean): Specifies whether integer variables are relaxable

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Integer variables are not relaxable.</td>
</tr>
<tr>
<td>1</td>
<td>(all) All integer variables are relaxable.</td>
</tr>
</tbody>
</table>

mip_restart (boolean): Specifies whether to enable the MIP restart procedure

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(off) Do not enable the MIP restart procedure.</td>
</tr>
<tr>
<td>1</td>
<td>(on) Enable the MIP restart procedure.</td>
</tr>
</tbody>
</table>

mip_rootalg (integer): Specifies which algorithm to use for the root node solve in MIP (same options as algorithm user option)

Range: {0, ..., 5}

Default: 0

mip_rounding (integer): Specifies the MIP rounding rule to apply

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro choose the rounding rule.</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>(none)  No rounding heuristic is used.</td>
</tr>
<tr>
<td>2</td>
<td>(heur_only) Round using a fast heuristic only.</td>
</tr>
<tr>
<td>3</td>
<td>(nlp_sometimes) Round and solve a subproblem if likely to succeed.</td>
</tr>
<tr>
<td>4</td>
<td>(nlp_always) Always round and solve a subproblem.</td>
</tr>
</tbody>
</table>

**mip_selectdir (boolean)**: Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch and bound tree

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(down) Choose the down (i.e. (&lt;=) node first.</td>
</tr>
<tr>
<td>1</td>
<td>(up) Choose the up (i.e. (&gt;=) node first.</td>
</tr>
</tbody>
</table>

**mip_selectrule (integer)**: Specifies the MIP select rule for choosing the next node in the branch and bound tree

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(auto) Let Knitro choose the node selection rule.</td>
</tr>
<tr>
<td>1</td>
<td>(depth_first) Search the tree using a depth first procedure.</td>
</tr>
<tr>
<td>2</td>
<td>(best_bound) Select the node with the best relaxation bound.</td>
</tr>
<tr>
<td>3</td>
<td>(combo_1) Use depth first unless pruned, then best bound.</td>
</tr>
</tbody>
</table>

**mip_strong_candlim (integer)**: Specifies the maximum number of candidates to explore for MIP strong branching

Range: \([-\infty, ..., \infty]\)

Default: 128

**mip_strong_level (integer)**: Specifies the maximum number of tree levels on which to perform MIP strong branching

Range: \([-\infty, ..., \infty]\)

Default: 10

**mip_strong_maxit (integer)**: Specifies the maximum number of iterations to allow for MIP strong branching solves

Range: \([-\infty, ..., \infty]\)

Default: 1000

**mip_terminate (integer)**: Specifies conditions for terminating the MIP algorithm

Default: 0
### Mip_Zerohalf

**mip_zerohalf (integer):** Specifies rules for adding zero-half cuts

- **Default:** 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Do not add zero-half cuts.</td>
</tr>
<tr>
<td>1</td>
<td>(root) Add cuts derived from equalities at the root node only.</td>
</tr>
<tr>
<td>2</td>
<td>(tree) Add cuts derived from equalities at every node depending on the solution of the relaxation and the cut generation strategy.</td>
</tr>
<tr>
<td>3</td>
<td>(all) Add cuts derived from equalities combining both root and tree cut generations.</td>
</tr>
</tbody>
</table>

### Ms_Deterministic

**ms_deterministic (boolean):** Indicates whether Knitro multi-start procedure will be deterministic (when ms_termine = 0)

- **Default:** 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) multithreaded multi-start is non-deterministic.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) multithreaded multi-start is deterministic (when ms_termine = 0).</td>
</tr>
</tbody>
</table>

### Ms_Enable

**ms_enable (boolean):** Indicates whether Knitro will solve from multiple start points to find a better local minimum

- **Default:** 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Knitro solves from a single initial point.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Knitro solves using multiple start points.</td>
</tr>
</tbody>
</table>

### Ms_Maxbndrange

**ms_maxbndrange (real):** Specifies the maximum range that an unbounded variable can take when determining new start points

Specifies the maximum range that an unbounded variable can take when determining new start points. If a variable is unbounded in one or both directions, then new start point values are restricted by the option. If $x_i$ is such a variable, then all initial values satisfy $\max\{l_i, x_i^0 - \text{ms_maxbndrange}/2\} \leq x_i \leq \min\{u_i, x_i^0 + \text{ms_maxbndrange}/2\}$, where $x_i^0$ is the initial value of $x_i$ provided by the user, and $l_i$ and $u_i$ are the variable bounds (possibly infinite) on $x_i$. This option has no effect unless ms_enable = yes.

- **Range:** $[-\infty, \infty]$

- **Default:** 1000

### Ms_Maxsolves

**ms_maxsolves (integer):** Specifies how many start points to try in multi-start

Specifies how many start points to try in multi-start. This option has no effect unless ms_enable = yes.

- **Default:** 0
Let Knitro automatically choose a value based on the problem size. The value is \( \min(200, 10 N) \), where \( N \) is the number of variables in the problem.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let Knitro automatically choose a value based on the problem size. The value is ( \min(200, 10 N) ), where ( N ) is the number of variables in the problem.</td>
</tr>
<tr>
<td>( n )</td>
<td>Try ( n &gt; 0 ) start points.</td>
</tr>
</tbody>
</table>

**ms_maxtime_cpu (real):** Specifies, in seconds, the maximum allowable CPU time before termination

Specifies, in seconds, the maximum allowable CPU time before termination. The limit applies to the operation of Knitro since multi-start began; in contrast, the value of *maxtime_cpu* limits how long Knitro iterates from a single start point. Therefore, *ms_maxtime_cpu* should be greater than *maxtime_cpu*. This option has no effect unless *ms_enable = yes*.

Range: \([-\infty, \infty]\)

Default: 100000000

**ms_maxtime_real (real):** Specifies, in seconds, the maximum allowable real time before termination

Specifies, in seconds, the maximum allowable real time before termination. The limit applies to the operation of Knitro since multi-start began; in contrast, the value of *maxtime_real* limits how long Knitro iterates from a single start point. Therefore, *ms_maxtime_real* should be greater than *maxtime_real*. This option has no effect unless *ms_enable = yes*.

Range: \([-\infty, \infty]\)

Default: 100000000

**ms_numthreads (integer):** Specify the number of threads to use for multi-start (when *ms_enable = 1*)

Specify the number of threads to use for multi-start (when *ms_enable = 1*). 0 Let Knitro choose the number of threads (currently sets *ms_numthreads* based on numthreads). \( n > 0 \) Use \( n \) threads for the multi-start (solve \( n \) problems in parallel).

Range: \( \{0, ..., \infty\} \)

Default: 0

**ms_num_to_save (integer):** Specifies the number of distinct feasible points to save in a file named knitro_mspoints

Specifies the number of distinct feasible points to save in a file named knitro_mspoints.log. Each point results from a Knitro solve from a different starting point, and must satisfy the absolute and relative feasibility tolerances. The file stores points in order from best objective to worst. Points are distinct if they differ in objective value or some component by the value of *ms_savetol* using a relative tolerance test. This option has no effect unless *ms_enable = yes*.

Range: \( \{0, ..., \infty\} \)

Default: 0

**ms_outsub (boolean):** Enable writing algorithm output to files for the parallel multi-start procedure

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not write detailed algorithm output to files.</td>
</tr>
<tr>
<td>1</td>
<td>Write detailed algorithm output to files named knitro_ms_*log.</td>
</tr>
</tbody>
</table>

**ms_savetol** (*real*): Specifies the tolerance for deciding if two feasible points are distinct

Specifies the tolerance for deciding if two feasible points are distinct. Points are distinct if they differ in objective value or some component by the value of *ms_savetol* using a relative tolerance test. A large value can cause the saved feasible points in the file knitro_mspoints.log to cluster around more widely separated points. This option has no effect unless *ms_enable* = yes. and *ms_num_to_save* is positive.

Range: $[-\infty, \infty]$

Default: $1e-06$

**ms_seed** (*integer*): Seed value used to generate random initial points in multi-start; should be a non-negative integer

Range: $\{0, ..., \infty\}$

Default: 0

**ms_startptrange** (*real*): Specifies the maximum range that each variable can take when determining new start points

Specifies the maximum range that each variable can take when determining new start points. If a variable has upper and lower bounds and the difference between them is less than or equal to *ms_startptrange*, then new start point values for the variable can be any number between its upper and lower bounds. If the variable is unbounded in one or both directions, or the difference between bounds is greater than *ms_startptrange*, then new start point values are restricted by the option. If $x_i$ is such a variable, then all initial values satisfy $\max\{b^L_i, x^0_i - \tau\} \leq x_i \leq \min\{b^U_i, x^0_i + \tau\}, \tau = \min\{ms\_startptrange/2, ms\_maxbndrange/2\}$ where $x^0_i$ is the initial value of $x_i$ provided by the user, and $b^L_i$ and $b^U_i$ are the variable bounds (possibly infinite) on $x_i$. This option has no effect unless *ms_enable* = yes.

Range: $[-\infty, \infty]$

Default: $1e+20$

**ms_terminate** (*integer*): Specifies the condition for terminating multi-start

Specifies the condition for terminating multi-start. This option has no effect unless *ms_enable* = yes.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terminate after <em>ms_maxsolves</em>.</td>
</tr>
<tr>
<td>1</td>
<td>Terminate after the first local optimal solution is found or <em>ms_maxsolves</em>, whichever comes first.</td>
</tr>
<tr>
<td>2</td>
<td>Terminate after the first feasible solution estimate is found or <em>ms_maxsolves</em>, whichever comes first.</td>
</tr>
<tr>
<td>3</td>
<td>Terminate after the first solution estimate of any type is found or <em>ms_maxsolves</em>, whichever comes first.</td>
</tr>
</tbody>
</table>
names (boolean): Enable to pass variable and equation names to Knitro

Default: 0

ncvx_qcqp_init (integer): Specifies the initialization strategy used for non-convex QPs and QCQPs

Specifies the initialization strategy used for non-convex QPs and QCQPs. In particular, these strategies may be more likely to cause Knitro to find global or better local solutions on non-convex quadratic programs (QPs) or non-convex quadratically constrained quadratic programs (QCQPs).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Knitro will automatically determine the strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(none) No special initialization strategy is used.</td>
</tr>
<tr>
<td>1</td>
<td>(linear) Initialize by solving a linear relaxation.</td>
</tr>
<tr>
<td>2</td>
<td>(hybrid) Initialize by solving a hybrid formulation.</td>
</tr>
<tr>
<td>3</td>
<td>(penalty) Initialize by solving a penalty formulation.</td>
</tr>
<tr>
<td>4</td>
<td>(cvxquad) Initialize by solving a convex quadratic relaxation.</td>
</tr>
</tbody>
</table>

newpoint (integer): Specifies additional action to take after every iteration in a solve of a continuous problem

Specifies additional action to take after every iteration in a solve of a continuous problem. An iteration of Knitro results in a new point that is closer to a solution. The new point includes values of x and Lagrange multipliers lambda. The "newpoint" feature in Knitro is currently only available for continuous problems (solved via KNsolve()).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Knitro takes no additional action.</td>
</tr>
<tr>
<td>1</td>
<td>(saveone) Knitro writes x and lambda to the file knitro_newpoint.log. Previous contents of the file are overwritten.</td>
</tr>
<tr>
<td>2</td>
<td>(saveall) Knitro appends x and lambda to the file knitro_newpoint.log. Warning: this option can generate a very large file. All iterates, including the start point, crossover points, and the final solution are saved.</td>
</tr>
</tbody>
</table>

objrange (real): Specifies the extreme limits of the objective function for purposes of determining unboundedness

Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function becomes greater than objrange for a feasible iterate, then the problem is determined to be unbounded and Knitro proceeds no further.

Range: [0, ∞]

Default: 1e+20

option_file (string): additional option file name - read only by KNITRO solver lib
**opttol** *(real)*: Specifies the final relative stopping tolerance for the KKT (optimality) error

Specifies the final relative stopping tolerance for the KKT (optimality) error. Smaller values of **opttol** result in a higher degree of accuracy in the solution with respect to optimality.

Range: \([0, \infty]\)

Default: \(1e-06\)

**opttolabs** *(real)*: Specifies the final absolute stopping tolerance for the KKT (optimality) error

Synonym: **opttol-abs**

Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of **opttolabs** result in a higher degree of accuracy in the solution with respect to optimality.

Range: \([0, \infty]\)

Default: \(0.001\)

**outappend** *(boolean)*: Specifies whether output should be started in a new file, or appended to existing files

Specifies whether output should be started in a new file, or appended to existing files. The option affects knitro.log. It does not affect knitro_newpoint.log, which is controlled by option newpoint.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Erase any existing files when opening for output.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Append output to any existing files.</td>
</tr>
</tbody>
</table>

**outdir** *(string)*: Specifies a single directory as the location to write all output files

Specifies a single directory as the location to write all output files. The option should be a full pathname to the directory, and the directory must already exist.

**outlev** *(integer)*: Controls the level of output produced by Knitro

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(none) Printing of all output is suppressed.</td>
</tr>
<tr>
<td>1</td>
<td>(summary) Print only summary information.</td>
</tr>
<tr>
<td>2</td>
<td>(iter,10) Print basic information every 10 iterations.</td>
</tr>
<tr>
<td>3</td>
<td>(iter) Print basic information at each iteration.</td>
</tr>
<tr>
<td>4</td>
<td>(iter_verbose) Print basic information and the function count at each iteration.</td>
</tr>
<tr>
<td>5</td>
<td>(iter,x) Print all the above, and the values of the solution vector x.</td>
</tr>
<tr>
<td>6</td>
<td>(all) Print all the above, and the values of the constraints c at x and the Lagrange multipliers lambda.</td>
</tr>
</tbody>
</table>

**outmode** *(integer)*: Specifies where to direct the output from Knitro

---
Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(screen) Output is directed to standard out (e.g., screen).</td>
</tr>
<tr>
<td>1</td>
<td>(file) Output is sent to a file named knitro.log.</td>
</tr>
<tr>
<td>2</td>
<td>(both) Output is directed to both the screen and file knitro.log.</td>
</tr>
</tbody>
</table>

**outname (string):** Use to specify a custom filename when output is written to a file using **outmode**

Default: 0

**output_time (boolean):** print output on where time is used

Default: 0

**out_csvinfo (boolean):** Controls whether or not to generates a file knitro_solve

Controls whether or not to generates a file knitro_solve.csv containing solve information in comma separated format.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) No solution information file is generated.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) The knitro_solve.csv solution file is generated.</td>
</tr>
</tbody>
</table>

**out_csvname (string):** Use to specify a custom csv filename when using **out_csvinfo**

**out_hints (boolean):** Specifies whether to print diagnostic hints (e.g. about user option settings) after solving

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not print any hints.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Print diagnostic hints on occasion.</td>
</tr>
</tbody>
</table>

**presolve (boolean):** Determine whether or not to use the Knitro presolver to try to simplify the model by removing variables or constraints

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) Do not use the Knitro presolver.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Enable the Knitro presolver.</td>
</tr>
</tbody>
</table>

**presolveop_tighten (integer):** Determine whether or not to enable the Knitro presolve operation to tighten variable bounds
Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Automatically determined (may depend on the algorithm).</td>
</tr>
<tr>
<td>0</td>
<td>(none) Do not tighten variable bounds (unless it removes a constraint).</td>
</tr>
<tr>
<td>1</td>
<td>(varbnd) Enable tightening variable bounds always.</td>
</tr>
</tbody>
</table>

**presolve_initpt** \((integer)\): Control whether the Knitro presolver can shift a user-supplied initial point

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose whether to allow shifting.</td>
</tr>
<tr>
<td>0</td>
<td>(noshift) Do not allow presolver to shift user-supplied initial point.</td>
</tr>
<tr>
<td>1</td>
<td>(linshift) Allow presolver to shift user-supplied initial point if it only appears in linear constraints.</td>
</tr>
<tr>
<td>2</td>
<td>(anyshift) Allow presolver to shift any user-supplied initial point.</td>
</tr>
</tbody>
</table>

**presolve_level** \((integer)\): Set the level of presolve operations to enable through the Knitro presolver

Set the level of presolve operations to enable through the Knitro presolver. A higher presolve level enables more complex presolve operations.

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>(auto) Let Knitro automatically choose the presolve level.</td>
</tr>
<tr>
<td>1</td>
<td>(level1) Enable the most basic presolve operations.</td>
</tr>
<tr>
<td>2</td>
<td>(level2) Enable more advanced presolve operations.</td>
</tr>
</tbody>
</table>

**presolve_passes** \((integer)\): Set a maximum limit on the number of passes through the Knitro presolve operations

Range: \(\{0, \ldots, \infty\}\)

Default: 10

**presolve_tol** \((real)\): Determines the tolerance used by the Knitro presolver to remove variables and constraints from the model

Determines the tolerance used by the Knitro presolver to remove variables and constraints from the model. If you believe the Knitro presolver is incorrectly modifying the model, use a smaller value for this tolerance (or turn the presolver off).

Range: \([0, \infty]\)

Default: \(1e-06\)

**qextractalg** \((integer)\): quadratic extraction algorithm in GAMS interface

Default: 0
restarts (integer): Specifies whether or not to enable automatic restarts in Knitro

Specifies whether or not to enable automatic restarts in Knitro. When enabled, if a Knitro algorithm seems to be converging slowly or not converging, the algorithm will automatically restart, which may help with convergence.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No automatic restarts allowed.</td>
</tr>
<tr>
<td>n</td>
<td>At most n&gt;0 automatic restarts may be performed.</td>
</tr>
</tbody>
</table>

restarts_maxit (integer): When restarts are enabled, this option can be used to specify a maximum number of iterations before enforcing a restart

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No iteration limit on restarts enforced.</td>
</tr>
<tr>
<td>n</td>
<td>At most n&gt;0 iterations are allowed without convergence before enforcing an automatic restart, if restarts are enabled.</td>
</tr>
</tbody>
</table>

scale (integer): Specifies whether to perform problem scaling of the objective function, constraint functions, or possibly variables

Specifies whether to perform problem scaling of the objective function, constraint functions, or possibly variables. If scaling is performed, internal computations, including some aspects of the optimality tests, are based on the scaled values, though the feasibility error is always computed in terms of the original, unscaled values.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) No scaling is performed.</td>
</tr>
<tr>
<td>1</td>
<td>(user_internal) User provided scaling is used if defined, otherwise Knitro internal scaling is applied.</td>
</tr>
<tr>
<td>2</td>
<td>(user_none) User provided scaling is used if defined, otherwise no scaling is applied.</td>
</tr>
<tr>
<td>3</td>
<td>(internal) Knitro internal scaling is applied.</td>
</tr>
</tbody>
</table>

soc (integer): Specifies whether or not to try second order corrections (SOC)

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for problems with highly nonlinear constraints.
Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) No second order correction steps are attempted.</td>
</tr>
<tr>
<td>1</td>
<td>(maybe) Second order correction steps may be attempted on some iterations.</td>
</tr>
<tr>
<td>2</td>
<td>(yes) Second order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.</td>
</tr>
</tbody>
</table>

strat_warm_start (boolean): Specifies whether or not to invoke a warm-start strategy

Specifies whether or not to invoke a warm-start strategy. A warm-start strategy may be beneficial when an initial point close to the solution can be provided. For example, this may occur when solving a sequence of closely related problems, and the solution from one problem can be used to initialize (or warm-start) the next problem in the sequence. The Knitro warm-start strategy will use this information to tune the algorithms to try to converge more quickly in this case. If the initial point is not sufficiently close to the solution, or is too infeasible, the warm-start strategy may not be helpful. This option is currently only used for the Knitro barrier/interior-point algorithms. In this case it may also be useful to experiment with different (smaller than default) values for the initial barrier parameter bar_initmu. In general, the closer the initial point is to the solution, the smaller this value should be (Knitro will try by default to initialize this to a good value when applying a warm-start strategy).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(no) No warm-start strategy is applied.</td>
</tr>
<tr>
<td>1</td>
<td>(yes) Knitro will apply a warm-start strategy with special tunings.</td>
</tr>
</tbody>
</table>

threads (integer): default thread count

Synonyms: gthreads numthreads

Controls the number of threads to use. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

By default, Knitro decides automatically how to use the threads specified. For example, if the multi-start method is enabled, it can run in parallel threads. If the multi-algorithm algorithm=5 is selected, the different algorithms can be run in parallel threads if threads are allocated. In case only one algorithm is running, multiple threads can be allocated to the MKL BLAS library or to the PARDISO linear solver via the threads option: see the blasoption, par_blasnumthreads, and par_lssnumthreads options for details.

Range: \{-\infty, \ldots, \infty\}

Default: GAMS threads

tuner (boolean): Indicates whether to invoke the Knitro-Tuner

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(off) Do not invoke the Knitro-Tuner.</td>
</tr>
<tr>
<td>1</td>
<td>(on) Invoke the Knitro-Tuner.</td>
</tr>
</tbody>
</table>
**tuner_maxtime_cpu (real):** Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner.

Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner. The limit applies to the operation of Knitro since the Knitro-Tuner began. In contrast, the value of `maxtime_cpu` places a time limit on each individual Knitro-Tuner solve for a particular option setting. Therefore, `tuner_maxtime_cpu` should be greater than `maxtime_cpu`. This option has no effect unless `tuner = on`.

Range: $[0, \infty]$  
Default: 100000000

**tuner_maxtime_real (real):** Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner.

Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner. The limit applies to the operation of Knitro since the Knitro-Tuner began. In contrast, the value of `maxtime_real` places a time limit on each individual Knitro-Tuner solve for a particular option setting. Therefore, `tuner_maxtime_real` should be greater than `maxtime_real`. This option has no effect unless `tuner = on`.

Range: $[0, \infty]$  
Default: 100000000

**tuner_optionsfile (string):** Can be used to specify the location of a Tuner options file.

**tuner_outsub (integer):** Enable writing additional Tuner subproblem solve output to files for the Knitro-Tuner procedure (tuner=1).

Default: 0  

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not write detailed solve output to files.</td>
</tr>
<tr>
<td>1</td>
<td>Write summary solve output to a file named knitro_tuner_summary.log.</td>
</tr>
<tr>
<td>2</td>
<td>Write detailed individual solve output to files named knitro_tuner_*_.log.</td>
</tr>
</tbody>
</table>

**tuner_terminate (integer):** Define the termination condition for the Knitro-Tuner procedure (tuner=1).

Default: 0  

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terminate after all solves have completed.</td>
</tr>
<tr>
<td>1</td>
<td>Terminate at first locally optimal solution.</td>
</tr>
<tr>
<td>2</td>
<td>Terminate at first feasible solution estimate.</td>
</tr>
<tr>
<td>3</td>
<td>Terminate at first solution estimate of any type.</td>
</tr>
</tbody>
</table>

**xtol (real):** The optimization process will terminate if the relative change in all components of the solution point estimate is less than `xtol` for `xtol_iters`.
The optimization process will terminate if the relative change in all components of the solution point estimate is less than `xtol` for `xtol_iters` consecutive iterations. If using the Interior/Direct or Interior/CG algorithm and the barrier parameter is still large, Knitro will first try decreasing the barrier parameter before terminating.

Range: \([0, \infty]\)

Default: \(1e-12\)

\(\text{xtol} \text{ } \text{integer}\): The optimization process will terminate if the relative change in the solution estimate is less than \(\text{xtol}\) for \(\text{xtol}\) consecutive iterations.

The optimization process will terminate if the relative change in the solution estimate is less than \(\text{xtol}\) for \(\text{xtol}\) consecutive iterations. If set to 0, Knitro chooses this value based on the solver and context. Currently Knitro sets this value to 3 unless the MISQP algorithm is being used, in which case the value is set to 1 by default.

Range: \([0, \ldots, \infty]\)

Default: 0

5.23.6 Knitro Termination Test and Optimality

5.23.6.1 Continuous problems

The first-order conditions for identifying a locally optimal solution of the problem (1) are:

\[
\nabla_x L(x, \lambda) = \nabla f(x) + \sum_{i=1}^{m} \lambda_i^c \nabla c_i(x) + \sum_{j=1}^{n} \lambda_j^b \nabla b_j(x) = 0
\]

\[
\lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] = 0, \quad i = 1 \ldots m
\]

\[
\lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] = 0, \quad j = 1 \ldots n
\]

\[
c_i^L \leq c_i(x) \leq c_i^U, \quad i = 1 \ldots m
\]

\[
b_j^L \leq x_j \leq b_j^U, \quad j = 1 \ldots n
\]

where \(\mathcal{I}\) and \(\mathcal{B}\) represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, \(\lambda_i^c\) is the Lagrange multiplier corresponding to constraint \(c_i(x)\), and \(\lambda_j^b\) is the Lagrange multiplier corresponding to the simple bounds on \(x_j\). There is exactly one Lagrange multiplier for each constraint and variable. The Lagrange multiplier may be restricted to take on a particular sign depending on the corresponding constraint or variable bounds.

In Knitro we define the feasibility error (\(\text{FeasErr}\)) at a point \(x^k\) to be the maximum violation of the constraints of (1), i.e.,

\[
\text{FeasErr} = \max_{i=1 \ldots m, j=1 \ldots n} (0, (c_i^L - c_i(x^k)), (c_i(x^k) - c_i^U), (b_j^L - x_j^k), (x_j^k - b_j^U))
\]

while the optimality error (\(\text{OptErr}\)) is defined as the maximum violation of the first three conditions of (3)

\[
\text{OptErr} = \max_{i=1 \ldots m, j=1 \ldots n} (\|\nabla_x L(x^k, \lambda^k)\|_{\infty}, \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))], \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)])
\]
The remaining conditions on the sign of the multipliers are enforced explicitly throughout the optimization. In order to take into account problem scaling in the termination test, the following scaling factors are defined:

\[
\tau_1 = \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), (x_j^0 - b_j^U)),
\]

\[
\tau_2 = \begin{cases} 
\max(1, \|\nabla f(x^k)\|_{\infty}), & \text{for constrained problems}, \\
\max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_{\infty})), & \text{for unconstrained problems}, 
\end{cases}
\]

where \(x^0\) represents the initial point. The special treatment for unconstrained problems is necessary, as for these problems, \(\|\nabla f(x^k)\|_{\infty} \to 0\) as a solution is approached, thus \(\max(1, \|\nabla f(x^k)\|_{\infty})\) would not be effective.

Knitro stops and declares **Locally optimal solution found** if the following stopping conditions are satisfied:

\[
\text{FeasErr} \leq \max(\tau_1 \ast \text{feastol, feastolabs}) \quad (4)
\]

\[
\text{OptErr} \leq \max(\tau_2 \ast \text{opttol, opttolabs}) \quad (5)
\]

where \(\text{feastol, opttol, feastolabs and opttolabs}\) are user-defined options (see section **Usage**).

This stopping test is designed to give the user much flexibility in deciding when the solution returned by Knitro is accurate enough. One can use a purely scaled stopping test (which is the recommended and default option) by setting \(\text{feastolabs}\) and \(\text{opttolabs}\) equal to \(0.0e0\). Likewise, an absolute stopping test can be enforced by setting \(\text{feastol}\) and \(\text{opttol}\) equal to \(0.0e0\).

**Unbounded problems**

Since by default Knitro uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied for an unbounded problem. For example, if \(\tau_2 \to \infty\) while the optimality error \(\text{OptErr}\) stays bounded, condition (5) will eventually be satisfied for some \(\text{opttol}>0\). If you suspect that your problem may be unbounded, using an absolute stopping test will allow Knitro to detect this.

### 5.23.6.2 Discrete problems

Algorithms for solving versions of (1) where one or more of the variables are restricted to take on only discrete values, proceed by solving a sequence of continuous relaxations, where the discrete variables are relaxed such that they can take on any continuous value. The *global* solutions \(f(x^g)\) of these relaxed problems provide a lower bound on the optimal objective value for problem (1) (upper bound if maximizing). If a feasible point is found for problem (1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1) (lower bound if maximizing). We will refer to these feasible points as *incumbent* points and denote the objective value at an incumbent point by \(f(x^I)\). Assuming all the continuous subproblems have been solved to global optimality (if the problem is convex, all local solutions are global solutions), an optimal solution of problem (1) is verified when the lower bound and upper bound are equal.

Knitro declares optimality for a discrete problem when the gap between the best (i.e., largest) lower bound \(f(x^g)\) and the best (i.e., smallest) upper bound \(f(x^I)\) is less than a threshold determined by the user options \(\text{mip\_opt\_gap\_abs}\) and \(\text{mip\_opt\_gap\_rel}\). Specifically, Knitro declares optimality when either

\[
f(x^I) - f(x^g) \leq \text{mip\_integral\_gap\_abs}
\]
or

\[ f(x_I) - f(x_R) \leq \text{mip\_integral\_gap\_rel} \cdot \max(1, |f(x_I)|), \]

where \text{mip\_opt\_gap\_abs} and \text{mip\_opt\_gap\_rel} are typically small positive numbers. Since these termination conditions assume that the continuous subproblems are solved to global optimality and Knitro only finds local solutions of nonconvex, continuous optimization problems, they are only reliable when solving convex, mixed integer problems. The integrality gap \( f(x_I) - f(x_R) \) should be non-negative although it may become slightly negative from roundoff error, or if the continuous subproblems are not solved to sufficient accuracy. If the integrality gap becomes largely negative, this may be an indication that the model is nonconvex, in which case Knitro may not converge to the optimal solution, and will be unable to verify optimality (even if it claims otherwise).

Note that the default values for \text{mip\_opt\_gap\_abs} and \text{mip\_opt\_gap\_rel} are taken from the GAMS options \text{optCA} and \text{optCR}, but an explicit setting of \text{mip\_opt\_gap\_abs} or \text{mip\_opt\_gap\_rel} will override those.

5.23.7 Knitro Output

If \text{outlev}=0 then all printing of output is suppressed. The description below assumes the default output level (\text{outlev}=2) except where indicated:

Nondefault Options:

This output lists all user options (see section Usage) which are different from their default values. If nothing is listed in this section then all user options are set to their default values.

Problem Characteristics:

The output begins with a description of the problem characteristics.

Iteration Information - Continuous Problems:

An iteration, in the context of Knitro, is defined as a step which generates a new solution estimate (i.e., a successful step). The columns of the iteration log are as follows:

- \text{Iter} Iteration number.
- \text{fCount} The cumulative number of function evaluations, only included if (\text{outlev}>3)
- \text{Objective} Gives the value of the objective function at the current iterate.
- \text{FeasErr} Gives a measure of the feasibility violation at the current iterate.
- \text{OptErr} Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate.
- ||\text{Step}|| The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).
- \text{CG its} The number of Projected Conjugate Gradient (CG) iterations required to compute the step.

If \text{outlev}=2, information is printed every 10 major iterations. If \text{outlev}=3 information is printed at each major iteration. If \text{outlev}>4 addition information is included in the log.

Iteration Information - Discrete Problems:

By default, the GAMS/Knitro link prints a log line at every 10th node. This frequency can be changed via the \text{mip\_outinterval} option. To turn off the node log completely, set the \text{mip\_outlevel} option to 0. The columns of the iteration log for discrete models are as follows:
- **Node** The node number. If an integer feasible point was found at a given node, it is marked with a *
- **Left** The current number of active nodes left in the branch and bound tree.
- **Iinf** The current number of active nodes left in the branch and bound tree.
- **Objective** Gives the value of the objective function at the solution of the relaxed subproblem solved at the current node. If the subproblem was infeasible or failed, this is indicated. Additional symbols may be printed at some nodes if the node was pruned (pr), integer feasible (f), or an integer feasible point was found through rounding (r).
- **Best relaxatn** The value of the current best relaxation (lower bound on the solution if minimizing).
- **Best incumbent** The value of the current best integer feasible point (upper bound on the solution if minimizing).

**Termination Message:** At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why the solver terminated. Below is a list of some possible termination messages.

- **EXIT: Locally optimal solution found.**
  Knitro found a locally optimal point which satisfies the stopping criterion (see section Knitro Termination Test and Optimality) for more detail on how this is defined. If the problem is convex (for example, a linear program), then this point corresponds to a globally optimal solution.

- **EXIT: Iteration limit reached.**
  The iteration limit was reached before being able to satisfy the required stopping criteria.

- **EXIT: Convergence to an infeasible point. Problem appears to be locally infeasible.**
  The algorithm has converged to an infeasible point from which it cannot further decrease the infeasibility measure. This happens when the problem is infeasible, but may also occur on occasion for feasible problems with nonlinear constraints or badly scaled problems. It is recommended to try various initial points. If this occurs for a variety of initial points, it is likely the problem is infeasible.

- **EXIT: Problem appears to be unbounded.**
  The objective function appears to be decreasing without bound, while satisfying the constraints.

- **EXIT: Current point cannot be improved.**
  No more progress can be made. If the current point is feasible it is likely it may be optimal, however the stopping tests cannot be satisfied perhaps because of degeneracy, ill-conditioning or bad scaling.

- **EXIT: Current point cannot be improved. Point appears to be optimal, but desired accuracy could not be achieved.**
  No more progress can be made, but the stopping tests are close to being satisfied (within a factor of 100) and so the current approximate solution is believed to be optimal.

- **EXIT: Time limit reached.**
  The time limit was reached before being able to satisfy the required stopping criteria.

- **EXIT: Evaluation error.**
  This termination value indicates that an evaluation error occurred (e.g., divide by 0, taking the square root of a negative number), preventing the optimization from continuing.

- **EXIT: Not enough memory available to solve problem.**
This termination value indicates that there was not enough memory available to solve the problem.

**Final Statistics:**

Following the termination message some final statistics on the run are printed. Both relative and absolute error values are printed.

**Solution Vector/Constraints:**

If `outlev=5`, the values of the solution vector are printed after the final statistics. If `outlev=6`, the final constraint values are also printed before the solution vector and the values of the Lagrange multipliers (or dual variables) are printed next to their corresponding constraint or bound.

### 5.23.8 Algorithm Options

#### 5.23.8.1 Automatic

By default, Knitro will automatically try to choose the best optimizer for the given problem based on the problem characteristics.

#### 5.23.8.2 Interior/Direct

If the Hessian of the Lagrangian is ill-conditioned or the problem does not have a large-dense Hessian, it may be advisable to compute a step by directly factoring the KKT (primal-dual) matrix rather than using an iterative approach to solve this system. Knitro offers the Interior/Direct optimizer which allows the algorithm to take direct steps by setting `algorithm=1`. This option will try to take a direct step at each iteration and will only fall back on the iterative step if the direct step is suspected to be of poor quality, or if negative curvature is detected.

Using the Interior/Direct optimizer may result in substantial improvements over Interior/CG when the problem is ill-conditioned (as evidenced by Interior/CG taking a large number of Conjugate Gradient iterations). We encourage the user to try both options as it is difficult to predict in advance which one will be more effective on a given problem. In each case, also experiment with the `bar_murule` option, as it is difficult to predict which update rule will work best.

**NOTE:** Since the Interior/Direct algorithm in Knitro requires the explicit storage of a Hessian matrix, this version can only be used with Hessian options, `hessopt=1, 2, 3` or `6`. It may not be used with Hessian options, `hessopt=4` or `5`, which only provide Hessian-vector products. Both the Interior/Direct and Interior/CG methods can be used with the `bar_feasible` option.

#### 5.23.8.3 Interior/CG

Since Knitro was designed with the idea of solving large problems, the Interior/CG optimizer in Knitro offers an iterative Conjugate Gradient approach to compute the step at each iteration. This approach has proven to be efficient in most cases and allows Knitro to handle problems with large, dense Hessians, since it does not require factorization of the Hessian matrix. The Interior/CG algorithm can be chosen by setting `algorithm=2`. It can use any of the Hessian options as well as the `bar_feasible` option.
5.23.8.4 Active Set

Knitro includes an active-set Sequential Linear-Quadratic Programming (SLQP) optimizer. This optimizer is particular advantageous when "warm starting" (i.e., when the user can provide a good initial solution estimate, for example, when solving a sequence of closely related problems). This algorithm is also the preferred algorithm for detecting infeasible problems quickly. The Active Set algorithm can be chosen by setting \texttt{algorithm=3}. It can use any of the Hessian options.

5.23.9 Other Knitro special features

This section describes in more detail some of the more important special features of Knitro and provides some guidance on how use them so that Knitro runs most efficiently for the problem at hand.

5.23.9.1 Second derivative options

The default version of Knitro assumes that exact second derivatives of the objective function and constraint functions can be computed. If this is possible and the cost of computing the second derivatives is not overly expensive, it is highly recommended to use exact second derivatives. However, Knitro also offers other options which are described in detail below.

**(Dense) Quasi-Newton BFGS**
The quasi-Newton BFGS option uses gradient information to compute a symmetric, \textit{positive-definite} approximation to the Hessian matrix. Typically this method requires more iterations to converge than the exact Hessian version. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in many cases. This option stores a \textit{dense} quasi-Newton Hessian approximation so it is only recommended for small to medium problems (\(n < 1000\)). The quasi-Newton BFGS option can be chosen by setting options value \texttt{hessopt=2}.

**(Dense) Quasi-Newton SR1**
As with the BFGS approach, the quasi-Newton SR1 approach builds an approximate Hessian using gradient information. However, unlike the BFGS approximation, the SR1 Hessian approximation is not restricted to be positive-definite. Therefore the quasi-Newton SR1 approximation may be a better approach, compared to the BFGS method, if there is a lot of negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case. The quasi-Newton SR1 approximation maintains a \textit{dense} Hessian approximation and so is only recommended for small to medium problems (\(n < 1000\)). The quasi-Newton SR1 option can be chosen by setting options value \texttt{hessopt=3}.

**Finite-difference Hessian-vector product option**
If the problem is large and gradient evaluations are not the dominate cost, then Knitro can internally compute Hessian-vector products using finite-differences. Each Hessian-vector product in this case requires one additional gradient evaluation. This option can be chosen by setting options value \texttt{hessopt=4}. This option is generally only recommended if the exact gradients are provided.

**NOTE:** This option may not be used when \texttt{algorithm=1}.

**Exact Hessian-vector products**
In some cases the problem which the user wishes to solve may have a large, dense Hessian which makes it impractical to store or work with the Hessian directly. The performance of this option should be nearly identical to the exact Hessian option but requires much less storage. This option can be chosen by setting options value \texttt{hessopt=5}.

**NOTE:** This option may not be used when \texttt{algorithm=1}.

**Limited-memory Quasi-Newton BFGS**
The limited-memory quasi-Newton BFGS option is similar to the dense quasi-Newton BFGS option described above. However, it is better suited for large-scale problems since, instead of storing a dense Hessian approximation, it only stores a limited number of gradient vectors used to approximate the Hessian. In general it requires more iterations to converge than the dense quasi-Newton BFGS approach but will be much more efficient on large-scale problems. This option can be chosen by setting options value \texttt{hessopt=6}. 

\[
\begin{align*}
\text{5.23 KNITRO} & \quad 1959
\end{align*}
\]
5.23.9.2 Feasible version

Knitro offers the user the option of forcing intermediate iterates to stay feasible with respect to the inequality constraints (it does not enforce feasibility with respect to the equality constraints however). Given an initial point which is sufficiently feasible with respect to all inequality constraints and selecting $\text{bar\_feasible} = 1$, forces all the iterates to strictly satisfy the inequality constraints throughout the solution process. For the feasible mode to become active the iterate $x$ must satisfy

$$cl + tol \leq c(x) \leq cu - tol$$

(21)

for all inequality constraints (i.e., for $cl \neq cu$). The tolerance $tol > 0$ by which an iterate must be strictly feasible for entering the feasible mode is determined by the parameter $\text{bar\_feasmodetol}$ which is $1.0e^{-4}$ by default. If the initial point does not satisfy (21) then the default infeasible version of Knitro will run until it obtains a point which is sufficiently feasible with respect to all the inequality constraints. At this point it will switch to the feasible version of Knitro and all subsequent iterates will be forced to satisfy the inequality constraints.

For a detailed description of the feasible version of Knitro see [32].

NOTE: This option may only be used when $\text{algorithm}=2$.

5.23.9.3 Honor Bounds

By default Knitro does not enforce that the simple bounds on the variables (1c) are satisfied throughout the optimization process. Rather, satisfaction of these bounds is only enforced at the solution. In some applications, however, the user may want to enforce that the initial point and all intermediate iterates satisfy the bounds $bl \leq x \leq bu$. This can be enforced by setting $\text{honorbnds}=1$.

5.23.9.4 Crossover

Interior-point (or barrier) methods are a powerful tool for solving large-scale optimization problems. However, one drawback of these methods is that they do not always provide a clear picture of which constraints are active at the solution. In general they return a less exact solution and less exact sensitivity information. For this reason, Knitro offers a crossover feature in which the interior-point method switches to the Active Set method at the interior-point solution estimate, in order to ”clean up” the solution and provide more exact sensitivity and active set information. The crossover procedure is controlled by the $\text{bar\_maxcrossit}$ option. If this option is greater than 0, then Knitro will attempt to perform $\text{bar\_maxcrossit}$ Active Set crossover iterations after the interior-point method has finished, to see if it can provide a more exact solution. This can be viewed as a form of post-processing. If $\text{bar\_maxcrossit}$ is not positive, then no crossover iterations are attempted.

The crossover procedure will not always succeed in obtaining a more exact solution compared with the interior-point solution. If crossover is unable to improve the solution within $\text{bar\_maxcrossit}$ crossover iterations, then it will restore the interior-point solution estimate and terminate. By default, Knitro will then print a message indicating that it was unable to improve the solution within the iterations allowed. In this case, you may want to increase the value of $\text{bar\_maxcrossit}$ and try again. If Knitro determines that the crossover procedure will not succeed, no matter how many iterations are tried, then a message of the form Crossover mode unable to improve solution. will be printed.

The extra cost of performing crossover is problem dependent. In most small or medium scale problems, the crossover cost is a small fraction of the total solve cost. In these cases it may be worth using the crossover procedure to obtain a more exact solution. On some large scale or difficult degenerate problems, however, the cost of performing crossover may be significant. It is recommended to experiment with this option to see whether improvement in the exactness of the solution is worth the additional cost.
5.23.9.5 Tuner

The Knitro-Tuner can help you identify some non-default options settings that may improve performance on a particular model or set of models. The Knitro tuner is enabled with the tuner option and controlled via the tuner family of options. If you are unsure about what Knitro options should be tuned to try to improve performance, you can run the default Knitro-Tuner by simply setting the option tuner=1 when solving with Knitro. This will cause Knitro to run your model with a variety of automatically determined option settings, and report some statistics at the end. Any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

If you have some ideas about which Knitro options you want to tune, you can tell Knitro which options you want it to tune, as well as specify the values for particular options that you want Knitro to explore. This can be done by specifying a Tuner options file. A Tuner options file is a simple text file that is similar to a standard Knitro options file, with some important differences:

- You can define multiple values (separated by spaces) for each option. This tells Knitro the values you want it to explore.
- You can specify an option name without any values. This will tell Knitro to explore all possible option values for that option. This only works for options that have a finite set of possible option value settings.
- A Tuner options file is loaded via the tuner_optionsfile option.

All possible combinations of options/values specified in a Tuner options file will be explored by Knitro, while any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

5.23.9.6 Solving Systems of Nonlinear Equations

Knitro is quite effective at solving systems of nonlinear equations. To solve a square system of nonlinear equations using Knitro one should specify the nonlinear equations as equality constraints (i.e., constraints with \( cl = cu \)), and specify the objective function (1a) as zero (i.e., \( f(x) = 0 \)).

5.23.9.7 Solving Least Squares Problems

There are two ways of using Knitro for solving problems in which the objective function is a sum of squares of the form

\[
f(x) = \frac{1}{2} \sum_{j=1}^{q} r_j(x)^2.\]

If the value of the objective function at the solution is not close to zero (the large residual case), the least squares structure of \( f \) can be ignored and the problem can be solved as any other optimization problem. Any of the Knitro options can be used.

On the other hand, if the optimal objective function value is expected to be small (small residual case) then Knitro can implement the Gauss-Newton or Levenberg-Marquardt methods which only require first derivatives of the residual functions, \( r_j(x) \), and yet converge rapidly. To do so, the user need only define the Hessian of \( f \) to be

\[
\nabla^2 f(x) = J(x)^T J(x),
\]

where

\[
J(x) = \begin{bmatrix}
\frac{\partial r_j}{\partial x_i} \\
\end{bmatrix} \quad j = 1, 2, \ldots, q \quad i = 1, 2, \ldots, n.
\]
The actual Hessian is given by

\[ \nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{q} r_j(x) \nabla^2 r_j(x); \]

the Gauss-Newton and Levenberg-Marquardt approaches consist of ignoring the last term in the Hessian.

Knitro will behave like a Gauss-Newton method by setting `algorithm=1`, and will be very similar to the classical Levenberg-Marquardt method when `algorithm=2`. For a discussion of these methods see, for example, [143].

## 5.24 LINDO and LINDOGlobal

### 5.24.1 Introduction

GAMS/LINDO finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDO supports most mathematical functions, including functions that are nonsmooth, such as `abs(x)` and or even discontinuous, such as `floor(x)`. Nonlinear solvers employing methods like successive linear programming (SLP) or generalized reduced gradient (GRG) return a local optimal solution to an NLP problem. However, many practical nonlinear models are non-convex and have more than one local optimal solution. In some applications, the user may want to find a global optimal solution.

The LINDO global optimization procedure (GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

GAMS/LINDO can automatically linearize a number of nonlinear relationships, such as `max(x,y)`, through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original nonlinear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and time it is also possible to disable the global solver and use GAMS/LINDO like a regular nonlinear solver.

GAMS/LINDO has a multistart feature that restarts the standard (non-global) nonlinear solver from a number of intelligently generated points. This allows the solver to find a number of locally optimal points and report the best one found. This alternative can be used when global optimization is costly. A user adjustable parameter controls the maximum number of multistarts to be performed.

LINDO automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDO, it will be solved as an LP at LP speed, regardless of what you said in the "solve using" statement. With the NLP parameter `NLP_QUADECHK` turned on, LINDO can detect hidden quadratic expressions and automatically recognize convex QCPs, as well as second-order cones (SOCP), like in Value-at-Risk models, allowing dramatically faster solution times via the barrier solver. When such models have integer variables, LINDO would use the barrier solver to solve all subproblems leading to significantly improved solution times when compared to the case with the standard NLP solver.
5.24.1.1 Licensing and software requirements

In order to use GAMS/LINDOGlobal, two licenses are required: a GAMS/LINDOGlobal license and a GAMS/CONOPT license. The additional CONOPT license requirement exists because LINDOGlobal uses CONOPT to solve the nonlinear subproblems. The GAMS/LINDOGlobal license places upper limits on the model size of 3,000 variables and 2,000 constraints.

To use GAMS/LINDO, only a GAMS/LINDO license is required. It imposes no upper limit on the model size and includes the capability to solve stochastic models (see section Stochastic Programming (SP) in GAMS/Lindo).

Neither the GAMS/LINDO nor the GAMS/LINDOGlobal license includes the Barrier solver option. The Barrier option is enabled via a separate license for the GAMS/MOSEK barrier solver.

5.24.1.2 Running GAMS/LINDO

GAMS/LINDO is capable of solving models of the following types: EMP (stochastic), LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP and MINLP. If GAMS/LINDO is not specified as the default solver for these models, it can be invoked by issuing one of the following commands before the solve statement:

```gams
option xxx=lindo;
option xxx=lindoglobal;
```

where `xxx` is one of: EMP, LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP, or MINLP.

You can also find global optima to math programs with equilibrium or complementarity constraints, type MPEC, by using the GAMS/NLPEC translator in conjunction with LINDO. You use NLPEC to translate complementarities into standard mathematical statements, e.g. \( h \cdot y = 0 \), and then use LINDO as the DNLP(Discontinuous Nonlinear) solver to solve the translated model. The following little GAMS model illustrates:

```gams
$title simple mpec example
variable f, x1, x2, y1, y2; positive
variable y1; y2.lo = -1; y2.up = 1;
equations cost, g, h1, h2;
cost..  f =E=  x1 + x2;
g..  sqr(x1) + sqr(x2) =L= 1;
h1..  x1 =G=  y1 - y2 + 1;
h2..  x2 + y2 =N= 0;
* declare h and y complementary
model example / cost, g, h1.y1, h2.y2 /;

option mpec=nlpec;
option dnlp=lindo;
solve example using mpec min f;
```

5.24.2 Supported nonlinear functions

GAMS/LINDO supports most nonlinear functions in global mode, including +, -, *, /, floor, modulo, sign, min, max, sqrt, exp, power, ln, log, sqrt, abs, cos, sin, tan, cosh, sinh, tanh, arccos, arcsin, arctan and logic expressions AND, OR, NOT, and IF. Be aware that using highly nonconvex functions may lead to long solve times.
5.24.3 Diagnosis of Infeasible or Unbounded Models

GAMS/LINDO offers two diagnostic tools, that can help users debug infeasible or unbounded optimization models. These tools can be called after the solver reports an infeasible or unbounded status for the model. When activating IIS Lindo finds an irreducible infeasible set (IIS) of constraints, whereas setting IUS, makes Lindo find an irreducible unbounded set (IUS) of variables. An IIS is a set of constraints that are infeasible taken together, but every strict subset is feasible. Similarly, an IUS is a set of unbounded variables such that fixing any one of them would make the model bounded. The IIS or IUS portion of the model will generally be much smaller than the original model. Thus, the user can track down formulation or data entry errors quickly. By isolating the source of infeasibility or unboundedness, the user can correct the model data such as right-hand side values, objective coefficients, senses of the constraints, and column bounds. Note that the IUS option is available for LPs only.

5.24.3.1 Infeasible Models

GAMS/Lindo's IIS option activates the IIS finder, after a model was tried to be solved and the solver returned a ”no feasible solution” message. For an LP, if an infeasible basis is not resident in the solver, the IIS finder cannot initiate the process to isolate an IIS. This can occur if the infeasibility is detected in the pre-solver before a basis is created, or the barrier solver has terminated without performing a basis crossover. To obtain an IIS for such cases, the pre-solve option will be turned off automatically and the model gets optimized again.

The constraints and bounds in the IIS are further classified into two disjoint sets: a necessary set and a sufficient set. The sufficient set refers to a crucial subset of the IIS in the sense that removing any one of its members from the entire model renders the model feasible. Note that not all infeasible models have sufficient sets. The necessary set contains those constraints and bounds that are likely to contribute to the overall infeasibility of the entire model. Thus, the necessary set requires a correction in at least one member to make the original model feasible. A constraint that has been marked as sufficient has a high probability of containing an error. In fact, if the model contains only one bad coefficient, the constraint containing it will be marked as sufficient.

To control the level of analysis when locating an IIS, one can set the option IIS_ANALYZE_LEVEL.

5.24.3.2 Unbounded Linear Programs

GAMS/Lindo's IUS option is similar to the IIS option, except that it is used to track down the source of an unbounded solution in a linear program. This tool analyzes the model and isolates an "irreducibly unbounded set" (IUS) of variables. As in the infeasibility case, the IUS is partitioned into sufficient and necessary sets to indicate the role of the member variables for the unboundedness of the overall model.

The variables in the sufficient set are crucial in the sense that fixing any of these variables makes the overall model bounded. However, fixing the variables in the necessary set does not ensure that there are no other sets of unbounded variables that cause unboundedness for the overall model.

To control the level of analysis when locating an IUS, one can set the option IUS_ANALYZE_LEVEL.
5.24.4 GAMS/LINDO output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDO's global solver.

LINDO 24Nov11 23.8.0 WIN 30200.30202 VS8 x86/MS Windows
LINDO Driver

Lindo API version 7.0.1.372 built on Nov 3 2011 21:49:01
Barrier Solver Version 6.0.0.114, Nonlinear Solver Version 3.15B
Platform Windows x86

Number of constraints: 3 le: 0, ge: 0, eq: 3, rn: 0 (ne:0)
Number of variables : 5 lb: 0, ub: 0, fr: 5, bx: 0 (fx:0)
Number of nonzeroes : 8 density=0.0053(%)  
Nonlinear variables : 5
Nonlinear constraints: 4
Nonlinear nonzeroes : 5+5

Starting global optimization ...

Number of nonlinear functions/operators: 3

Starting GOP presolve ...
First Call Local Solver
Find local solution, objvalue = 27.871905

Pre-check unboundedness
Computing reduced bound...
Searching for a better solution...

Starting reformulation ...

Model Input Operation Atomic Convex
Number of variables : 5 6 20 20
Number of constraints: 3 4 18 46
integer variables : 0 0 0 0
nonlinear variables : 5 5 9 0

Starting global search ...
Initial upper bound on objective: +2.931083e-002
Initial lower bound on objective: -3.167052e+022

<table>
<thead>
<tr>
<th>#NODEs</th>
<th>BOXES</th>
<th>LOWER BOUND</th>
<th>UPPER BOUND</th>
<th>RGAP</th>
<th>TIME(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-3.167052e+022</td>
<td>+2.931083e-002</td>
<td>1.0e+000</td>
<td>0 (*N)</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>-1.848574e+001</td>
<td>+2.931083e-002</td>
<td>2.1e-001</td>
<td>0 (*I)</td>
</tr>
<tr>
<td>22</td>
<td>21</td>
<td>+2.416053e-003</td>
<td>+2.931083e-002</td>
<td>2.7e-002</td>
<td>0 (*F)</td>
</tr>
</tbody>
</table>
Terminating global search ...

Global optimum found

Objective value : 0.0293108307216
Best Bound : 0.00241605257558
Factors (ok,stb) : 522 (100.00, 99.81)
Simplex iterations : 2503
Barrier iterations : 0
Nonlinear iterations : 433
Box iterations : 23
Total number of boxes : 21
Max. Depth : 5
First solution time (sec.) : 0
Best solution time (sec.) : 0
Total time (sec.) : 0

After determining the different kinds of nonlinear operators LINDO tries to linearize these within the presolving. When a feasible starting point is found the optimization starts and the log provides information about the progress. At the end it is reported if an optimum could be found and then the results as well as the used resources are summarized.

The following flags can be seen in the progress log:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*FP)</td>
<td>found a new MIP solution with feasibility pump</td>
</tr>
<tr>
<td>(*SBB)</td>
<td>found a new MIP solution in tree reorder</td>
</tr>
<tr>
<td>(*SE)</td>
<td>found a new MIP solution in simple enumeration</td>
</tr>
<tr>
<td>(*AB)</td>
<td>found a new MIP solution in advanced branching</td>
</tr>
<tr>
<td>(*AH)</td>
<td>found a new MIP solution with advanced heuristics</td>
</tr>
<tr>
<td>(*C)</td>
<td>found a new MIP solution after cuts added</td>
</tr>
<tr>
<td>(*T)</td>
<td>found a new MIP solution on the top</td>
</tr>
<tr>
<td>(*SRH)</td>
<td>found a new MIP solution in simple rounding heuristics</td>
</tr>
<tr>
<td>(*SB)</td>
<td>found a new MIP solution in strong branching</td>
</tr>
<tr>
<td>(*K)</td>
<td>found a new MIP solution in knapsack enumerator</td>
</tr>
<tr>
<td>(*)</td>
<td>found a new MIP solution normal branching</td>
</tr>
<tr>
<td>(??-)</td>
<td>found a new MIP solution with advanced heuristics (level$&gt;$10)</td>
</tr>
<tr>
<td>(*N)</td>
<td>found a new incumbent GOP solution</td>
</tr>
<tr>
<td>(*I)</td>
<td>stored a box with the incumbent solution into the GOP solution list</td>
</tr>
<tr>
<td>(*F)</td>
<td>determined the final GOP status</td>
</tr>
</tbody>
</table>

5.24.5 The GAMS/LINDO Options

GAMS/LINDO offers a diverse range of user-adjustable parameters to control the behavior of its solvers. While the default values of these parameters work best for most purposes, there may be cases the users prefer to work with different settings for a subset of the available parameters. This section gives a list of available GAMS/LINDO parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.
5.24.5.1 GAMS/LINDO Options File

In order to set GAMS/LINDO options, you need to set up an option file `lindo.opt` or `lindoglobal.opt` in your GAMS project directory. You must indicate in the model that you want to use the option file by inserting before the solve statement, the line:

```plaintext
<modelname>.optfile = 1;
```

where

```plaintext
<modelname>
```

is the name of the model referenced in the model statement. The option file is in plain text format containing a single GAMS/LINDO option per line. Each option identifier is followed by its target value with space or tab characters separating them. The lines starting with `*` character are treated as comments.

A sample option file `lindo.opt` looks like below

```plaintext
* Use(1) or Disable(0) global optimization for NLP/MINLP models
USEGOP 0

* Enable Multistart NLP solver
NLP_SOLVER 9

* Allow a maximum of 3 multistart attempts
NLP_MAXLOCALSEARCH 3

* Set an overall time limit of 200 secs.
SOLVER_TIMLMT 200
```

5.24.6 Summary of GAMS/Lindo Options

5.24.6.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECOMPOSITION_TYPE</td>
<td>decomposition to be performed on a linear or mixed integer model</td>
<td>1</td>
</tr>
<tr>
<td>FIND_BLOCK</td>
<td>graph partitioning method to find block structures</td>
<td>0</td>
</tr>
<tr>
<td>FIND_SYMMETRY_LEVEL</td>
<td>specifies the symmetry finding level</td>
<td>-1</td>
</tr>
<tr>
<td>FIND_SYMMETRY_PRINT_LEVEL</td>
<td>specifies print level for symmetry finding</td>
<td>0</td>
</tr>
<tr>
<td>INSTRUCT_SUBOUT</td>
<td>flag to specify how to deal with fixed variables in the instruction list</td>
<td>-1</td>
</tr>
<tr>
<td>MULTITHREAD_MODE</td>
<td>threading mode</td>
<td>-1</td>
</tr>
<tr>
<td>NUM_THREADS</td>
<td>number of parallel threads to be used</td>
<td>GAMS Threads</td>
</tr>
<tr>
<td>PROFILER_LEVEL</td>
<td>specifies the profiler level to break down the total cpu time into.</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_CONCURRENT_OPTMODE</td>
<td>controls if simplex and interior-point optimizers will run concurrently</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SOLVER_CUTOFFVAL</td>
<td>solver will exit if optimal solution is worse than this</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_FEASTOL</td>
<td>feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>SOLVER_IPMSOL</td>
<td>basis crossover flag for barrier solver</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_IUSOL</td>
<td>flag for computing basic solution for infeasible model</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_METHOD</td>
<td>specifies the method to use when generic solver is invoked</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_MODE</td>
<td>controls some of the advanced strategies when solving LPs</td>
<td>1</td>
</tr>
<tr>
<td>SOLVER_OPTTOL</td>
<td>dual feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>SOLVER_PRE_ELIM_FILL</td>
<td>fill-in introduced by the eliminations during presolve</td>
<td>1000</td>
</tr>
<tr>
<td>SOLVER_RESTART</td>
<td>starting basis flag</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_TIMLMT</td>
<td>time limit in seconds for continous solver</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>SOLVER_USECUTOFFVAL</td>
<td>flag for using cutoff value</td>
<td>0</td>
</tr>
<tr>
<td>TUNER_PRINT_LEVEL</td>
<td>specifies the amount of print to do during parameter tuning</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.24.6.2 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP_AIJ_ZEROTOL</td>
<td>coefficient matrix zero tolerance</td>
<td>2.22045e-16</td>
</tr>
<tr>
<td>LP_BIGM</td>
<td>big-M for phase-I</td>
<td>1e6</td>
</tr>
<tr>
<td>LP_BNDINF</td>
<td>big-M to truncate lower and upper bounds in single phase dual-simplex</td>
<td>1e15</td>
</tr>
<tr>
<td>LP_DPSWITCH</td>
<td>specifies whether LP primal-dual simplex switch is enabled or not</td>
<td>1</td>
</tr>
<tr>
<td>LP_DRATIO</td>
<td>controls the dual min-ratio strategy</td>
<td>1</td>
</tr>
<tr>
<td>LP_DYNOBJFACT</td>
<td>Dynamic obj factor</td>
<td>0.75</td>
</tr>
<tr>
<td>LP_DYNOBJMODE</td>
<td>Dynamic obj mode</td>
<td>0</td>
</tr>
<tr>
<td>LP_ITRLMT</td>
<td>simplex iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>LP_PIV_BIGTOL</td>
<td>simplex maximum pivot tolerance</td>
<td>1e-5</td>
</tr>
<tr>
<td>LP_PIV_ZEROTOL</td>
<td>simplex pivot zero tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>LP_PPARTIAL</td>
<td>primal simplex partial pricing method</td>
<td>0</td>
</tr>
<tr>
<td>LP_PRELEVEL</td>
<td>controls the amount and type of LP pre-solving</td>
<td>126</td>
</tr>
<tr>
<td>LP_RATRANGE</td>
<td>controls the number of pivot-candidates to consider when searching for a stable pivot in LU decomposition</td>
<td>4</td>
</tr>
<tr>
<td>LP_SCALE</td>
<td>scaling flag</td>
<td>1</td>
</tr>
<tr>
<td>LP_SPRINT_COLFACT</td>
<td>maximum number of columns in Sprint as a factor of number of rows</td>
<td>10</td>
</tr>
<tr>
<td>LP_SPRINT_MAXPASS</td>
<td>maximum number of passes in Sprint method</td>
<td>100</td>
</tr>
<tr>
<td>LP_SPRINT_SUB</td>
<td>LP method for subproblem in Sprint method</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.24.6.3 IPM Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPM_BASIS_REL_TOL_S</td>
<td>maximum relative dual bound violation allowed in an optimal basic solution</td>
<td>1e-12</td>
</tr>
<tr>
<td>IPM_BASIS_TOL_S</td>
<td>maximum absolute dual bound violation in an optimal basic solution</td>
<td>1e-7</td>
</tr>
<tr>
<td>IPM_BASIS_TOL_X</td>
<td>maximum absolute primal bound violation allowed in an optimal basic solution</td>
<td>1e-7</td>
</tr>
<tr>
<td>IPM_BI_LU_TOL_REL_PIV</td>
<td>relative pivot tolerance used in the LU factorization in the basis identification procedure</td>
<td>1e-2</td>
</tr>
<tr>
<td>IPM_CHECK_CONVEXITY</td>
<td>flag to check convexity of a quadratic program using barrier solver</td>
<td>1</td>
</tr>
<tr>
<td>IPM_CO_TOL_INFEAS</td>
<td>maximum bound infeasibility tolerance for Conic solver</td>
<td>1e-10</td>
</tr>
<tr>
<td>IPM_MAX_ITERATIONS</td>
<td>ipm iteration limit</td>
<td>1000</td>
</tr>
<tr>
<td>IPM_NUM_THREADS</td>
<td>number of threads to run the interiorpoint optimizer on</td>
<td>1</td>
</tr>
<tr>
<td>IPM_OFF_COL_TRH</td>
<td>extent for detecting the offending columns in the Jacobian of the constraint matrix</td>
<td>40</td>
</tr>
<tr>
<td>IPM_TOL_DFEAS</td>
<td>dual feasibility tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_DSAFE</td>
<td>controls the initial dual starting point</td>
<td>1</td>
</tr>
<tr>
<td>IPM_TOL_INFEAS</td>
<td>infeasibility tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td>IPM_TOL_MU_RED</td>
<td>relative complementarity gap tolerance</td>
<td>1e-16</td>
</tr>
<tr>
<td>IPM_TOL_PATH</td>
<td>how close to follow the central path</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_PFEAS</td>
<td>primal feasibility tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_PSAFE</td>
<td>controls the initial primal starting point</td>
<td>1</td>
</tr>
<tr>
<td>IPM_TOL_REL_STEP</td>
<td>relative step size to the boundary</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

### 5.24.6.4 MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIP_ABSCUTTOL</td>
<td>MIP absolute cut tolerance</td>
<td>-1.0</td>
</tr>
<tr>
<td>MIP_ABSOPTTOL</td>
<td>MIP absolute optimality tolerance</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>MIP_ADDCUTOBJTOL</td>
<td>required objective improvement to continue generating cuts</td>
<td>1.5625e-5</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>MIP_ADDCUTPER</td>
<td>percentage of constraint cuts that can be added</td>
<td>0.75</td>
</tr>
<tr>
<td>MIP_ADDCUTPER_TREE</td>
<td>percentage of constraint cuts that can be added at child nodes</td>
<td>0.5</td>
</tr>
<tr>
<td>MIP_AGCUTLIM_TOP</td>
<td>max number of constraints involved in derivation of aggregation cut at root node</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_AGCUTLIM_TREE</td>
<td>max number of constraints involved in derivation of aggregation cut at tree nodes</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_ANODES_SWITCH_DF</td>
<td>threshold on active nodes for switching to depth-first search</td>
<td>50000</td>
</tr>
<tr>
<td>MIP_AOPTTIMLIM</td>
<td>time in seconds beyond which the relative optimality tolerance will be applied</td>
<td>100</td>
</tr>
<tr>
<td>MIP_BIGM_FOR_INTTOL</td>
<td>threshold for which coefficient of a binary variable would be considered as big-M</td>
<td>1e8</td>
</tr>
<tr>
<td>MIP_BRANCHDIR</td>
<td>first branching direction</td>
<td>0</td>
</tr>
<tr>
<td>MIP_BRANCHRULE</td>
<td>rule for choosing the variable to branch</td>
<td>0</td>
</tr>
<tr>
<td>MIP_BRANCHLIMIT</td>
<td>limit on the total number of branches to be created during branch and bound</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_BRANCHPRIOR</td>
<td>controls how variable selection priorities are set and used</td>
<td>0</td>
</tr>
<tr>
<td>MIP_CUTDEPTH</td>
<td>threshold value for the depth of nodes in the branch and bound tree</td>
<td>8</td>
</tr>
<tr>
<td>MIP_CUTFREQ</td>
<td>frequency of invoking cut generation at child nodes</td>
<td>10</td>
</tr>
<tr>
<td>MIP_CUTLEVEL_TOP</td>
<td>combination of cut types to try at the root node when solving a MIP</td>
<td>57342</td>
</tr>
<tr>
<td>MIP_CUTLEVEL_TREE</td>
<td>combination of cut types to try at child nodes when solving a MIP</td>
<td>53246</td>
</tr>
<tr>
<td>MIP_CUTOFFOBJ</td>
<td>defines limit for branch and bound</td>
<td>1e30</td>
</tr>
<tr>
<td>MIP_CUTTIMLIM</td>
<td>time to be spent in cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_DELTA</td>
<td>near-zero value used in linearizing nonlinear expressions</td>
<td>1e-6</td>
</tr>
<tr>
<td>MIP_DUAL_SOLUTION</td>
<td>flag for computing dual solution of LP relaxation</td>
<td>0</td>
</tr>
<tr>
<td>MIP_FP_ITRLIM</td>
<td>iteration limit for feasibility pump heuristic</td>
<td>500</td>
</tr>
<tr>
<td>MIP_FP_MODE</td>
<td>mode for the feasibility pump heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_FP_OPT_METHOD</td>
<td>optimization and reoptimization method for feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>MIP_FP_TIMLIM</td>
<td>time limit for feasibility pump heuristic</td>
<td>1800</td>
</tr>
<tr>
<td>MIP_FP_WEIGHT</td>
<td>weight of the objective function in the feasibility pump</td>
<td>1</td>
</tr>
<tr>
<td>MIP_GENERAL_MODE</td>
<td>general strategy in solving MIPs</td>
<td>0</td>
</tr>
<tr>
<td>MIP_HEUELEVEL</td>
<td>specifies heuristic used to find integer solution</td>
<td>3</td>
</tr>
<tr>
<td>MIP_HEUMINTIMLIM</td>
<td>minimum time in seconds to be spent in finding heuristic solutions</td>
<td>0</td>
</tr>
<tr>
<td>MIP_HEU_DROP_OBJ</td>
<td>flag for whether to use without OBJ heuristic</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>MIP_HEU_MODE</td>
<td>heuristic used in MIP solver</td>
<td>0</td>
</tr>
<tr>
<td>MIP_INTTOL</td>
<td>absolute integer feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>MIP_ITRLIM</td>
<td>iteration limit for branch and bound</td>
<td>infinity</td>
</tr>
<tr>
<td>MIP_KBEST_USE_GOP</td>
<td>specifies whether to use gop solver in MIP KBest</td>
<td>0</td>
</tr>
<tr>
<td>MIP_KEEPINMEM</td>
<td>flag for keeping LP bases in memory</td>
<td>1</td>
</tr>
<tr>
<td>MIP_LBIGM</td>
<td>Big-M value used in linearizing nonlinear expressions</td>
<td>10000</td>
</tr>
<tr>
<td>MIP_LSOLTIMLIM</td>
<td>time limit until finding a new integer solution</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_MAKECUT_INACTIVE_COUNT</td>
<td>threshold for times a cut could remain active after successive reoptimization</td>
<td>10</td>
</tr>
<tr>
<td>MIP_MAXCUTPASS_TOP</td>
<td>number passes to generate cuts on the root node</td>
<td>200</td>
</tr>
<tr>
<td>MIP_MAXCUTPASS_TREE</td>
<td>number passes to generate cuts on the child nodes</td>
<td>2</td>
</tr>
<tr>
<td>MIP_MAXNONIMP_CUTPASS</td>
<td>number of passes allowed in cut-generation that does not improve current relaxation</td>
<td>3</td>
</tr>
<tr>
<td>MIP_MAXNUM_MIP_SOL_STORAGE</td>
<td>maximum number of k-best solutions to store</td>
<td>1</td>
</tr>
<tr>
<td>MIP_MINABSOBJSTEP</td>
<td>value to update cutoff value each time a mixed integer solution is found</td>
<td>0</td>
</tr>
<tr>
<td>MIP_NODESELRULE</td>
<td>specifies the node selection rule</td>
<td>0</td>
</tr>
<tr>
<td>MIP_NUM_THREADS</td>
<td>number of parallel threads to use by the parallel MIP solver</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_FP</td>
<td>flag for whether to use parallelization on the feasibility pump heuristic</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_FP_MODE</td>
<td>flag for the mode of parallel feasibility pump</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PARA_INIT_NODE</td>
<td>number of initial nodes for MIP parallelization</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_PARA_ITR_MODE</td>
<td>flag for iteration mode in MIP parallelization</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_RND_ITRLMT</td>
<td>iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations</td>
<td>2.0</td>
</tr>
<tr>
<td>MIP_PARA_SUB</td>
<td>flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PEROPTTOL</td>
<td>MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds</td>
<td>1e-5</td>
</tr>
<tr>
<td>MIP_PERSPECTIVEREFORM</td>
<td>flag for whether to use Perspective Reformulation</td>
<td>1</td>
</tr>
<tr>
<td>MIP_POLISH_ALPHA_TARGET</td>
<td>proportion solutions in the pool to initiate a polishing-task at the current node</td>
<td>0.6</td>
</tr>
<tr>
<td>MIP_POLISH_MAX_BRANCH_COUNT</td>
<td>maximum number of branches to polish</td>
<td>2000</td>
</tr>
<tr>
<td>MIP_POLISH_NUM_BRANCH_NEXT</td>
<td>number of branches to polish in the next round</td>
<td>4000</td>
</tr>
<tr>
<td>MIP_PREHEU_DFE_VSTLIM</td>
<td>limit for the variable visit in depth first enumeration</td>
<td>200</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>MIP_PREHEU_LEVEL</td>
<td>heuristic level for the prerelax solver</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PREHEU_TC_ITERLIM</td>
<td>iteration limit for the two change heuristic</td>
<td>30000000</td>
</tr>
<tr>
<td>MIP_PREHEU_VAR_SEQ</td>
<td>sequence of the variable considered by the prerelax heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_PRELEVEL</td>
<td>controls the amount and type of MIP presolving at root node</td>
<td>3070</td>
</tr>
<tr>
<td>MIP_PRELEVEL_TREE</td>
<td>amount and type of MIP presolving at tree nodes</td>
<td>1214</td>
</tr>
<tr>
<td>MIP_PRE_ELIM_FILL</td>
<td>controls fill-in introduced by eliminations during pre-solve</td>
<td>100</td>
</tr>
<tr>
<td>MIP_PSEUDOCOST_RULE</td>
<td>specifies the rule in pseudocost computations for variable selection</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PSEUDOCOST_WEIGT</td>
<td>weight in pseudocost computations for variable selection</td>
<td>1.5625e-05</td>
</tr>
<tr>
<td>MIP_REDCOSTFIX_CUTOFF</td>
<td>cutoff value as a percentage of the reduced costs</td>
<td>0.9</td>
</tr>
<tr>
<td>MIP_REDCOSTFIX_CUTOFF_TREE</td>
<td>cutoff value as a percentage of the reduced costs at tree nodes</td>
<td>0.9</td>
</tr>
<tr>
<td>MIP_RELINTTOL</td>
<td>relative integer feasibility tolerance</td>
<td>8e-6</td>
</tr>
<tr>
<td>MIP_RELOPTTOL</td>
<td>MIP relative optimality tolerance</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>MIP_REOPT</td>
<td>optimization method to use when doing reoptimization</td>
<td>0</td>
</tr>
<tr>
<td>MIP_SCALING_BOUND</td>
<td>maximum difference between bounds of an integer variable for enabling scaling</td>
<td>10000</td>
</tr>
<tr>
<td>MIP_SOLLIM</td>
<td>integer solution limit for MIP solver</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_SOLVERTYPE</td>
<td>optimization method to use when solving mixed-integer models</td>
<td>0</td>
</tr>
<tr>
<td>MIP_STRONGBRANCHDONUM</td>
<td>minimum number of variables to try the strong branching on</td>
<td>3</td>
</tr>
<tr>
<td>MIP_STRONGBRANCHLEVEL</td>
<td>depth from the root in which strong branching is used</td>
<td>10</td>
</tr>
<tr>
<td>MIP_SWITCHFAC_SIM_IPM_TIME</td>
<td>factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_SYMMETRY_MODE</td>
<td>specifies mip symmetry handling methods</td>
<td>0</td>
</tr>
<tr>
<td>MIP_TIMLIM</td>
<td>time limit in seconds for integer solver</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>MIP_TOPOPT</td>
<td>optimization method to use when there is no previous basis</td>
<td>0</td>
</tr>
<tr>
<td>MIP_TREEREORDERLEVEL</td>
<td>tree reordering level</td>
<td>10</td>
</tr>
<tr>
<td>MIP_TREEREORDERMODE</td>
<td>tree reordering mode</td>
<td>1</td>
</tr>
<tr>
<td>MIP_USECUTOFFOBJ</td>
<td>flag for using branch and bound limit</td>
<td>1</td>
</tr>
<tr>
<td>MIP_USE_CUTS_HEU</td>
<td>controls if cut generation is enabled during MIP heuristics</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_USE_ENUM_HEU</td>
<td>frequency of enumeration heuristic</td>
<td>4</td>
</tr>
<tr>
<td>MIP_USE_INT_ZERO_TOL</td>
<td>controls if all MIP calculations would be based on absolute integer feasibility tolerance</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.24.6.5 NLP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP_AUTODERIV</td>
<td>defining type of computing derivatives</td>
<td>0</td>
</tr>
<tr>
<td>NLP_AUTOHESS</td>
<td>flag for using Second Order Automatic Differentiation for solving NLP</td>
<td>0</td>
</tr>
<tr>
<td>NLP_CONIC_REFORM</td>
<td>determines if to explore conic reformulation</td>
<td>1</td>
</tr>
<tr>
<td>NLP_CUTOFFOBJ</td>
<td>as soon as any multi-start thread achieves this value all threads stop</td>
<td>-1e30</td>
</tr>
<tr>
<td>NLP_DERIV_DIFFTYPE</td>
<td>flag indicating the technique used in computing derivatives with finite differences</td>
<td>0</td>
</tr>
<tr>
<td>NLP_FEASCHK</td>
<td>how to report results when solution satisfies tolerance of scaled but not original model</td>
<td>1</td>
</tr>
<tr>
<td>NLP_FEASTOL</td>
<td>feasibility tolerance for nonlinear constraints</td>
<td>1e-6</td>
</tr>
<tr>
<td>NLP_INF</td>
<td>numeric infinity for nonlinear models</td>
<td>1e30</td>
</tr>
<tr>
<td>NLP_IPM2GRG</td>
<td>switch from IPM solver to GRG solver when IPM fails due to numerical errors</td>
<td>1</td>
</tr>
<tr>
<td>NLP_ITERS_PER_LOGLINE</td>
<td>number of nonlinear iterations to elapse before next progress message</td>
<td>10</td>
</tr>
<tr>
<td>NLP_JTRLMT</td>
<td>nonlinear iteration limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>NLP_LINEARZ</td>
<td>extent to which the solver will attempt to linearize nonlinear models</td>
<td>0</td>
</tr>
<tr>
<td>NLP_LINEARZ_WB_CONSISTENT</td>
<td>determines if linearization process is consistent with WB/excel calculation</td>
<td>0</td>
</tr>
<tr>
<td>NLP_MAXLOCALSEARCH</td>
<td>maximum number of local searches</td>
<td>5</td>
</tr>
<tr>
<td>NLP_MAXLOCALSEARCH_TREE</td>
<td>maximum number of multistarts</td>
<td>1</td>
</tr>
<tr>
<td>NLP_MAX_RETRY</td>
<td>maximum number refinement retries to purify the final NLP solution</td>
<td>5</td>
</tr>
<tr>
<td>NLP_MAXRETRY</td>
<td>maximum number refinement retries to purify the final NLP solution</td>
<td>5</td>
</tr>
<tr>
<td>NLP_MSW_EUCDIST_THRES</td>
<td>euclidean distance threshold in multistart search</td>
<td>0.001</td>
</tr>
<tr>
<td>NLP_MSW_FILTMODE</td>
<td>filtering mode to exclude certain domains during sampling in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXNOIMP</td>
<td>maximum number of consecutive populations to generate without any improvements</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXPOP</td>
<td>maximum number of populations to generate in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXREF</td>
<td>maximum number of reference points to generate trial points in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_NORM</td>
<td>norm to measure the distance between two points in multistart search</td>
<td>2</td>
</tr>
<tr>
<td>NLP_MSW_NUM_THREADS</td>
<td>number of parallel threads to be used when solving an NLP model with the multistart solver</td>
<td>1</td>
</tr>
<tr>
<td>NLP_MSW_OVERLAP_RATIO</td>
<td>rate of replacement in successive populations</td>
<td>0.1</td>
</tr>
<tr>
<td>NLP_MSW_POXDIST_THRES</td>
<td>penalty function neighborhood threshold in multistart search</td>
<td>0.01</td>
</tr>
<tr>
<td>NLP_MSW_PREPMODE</td>
<td>preprocessing strategies in multistart solver</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_RG_SEED</td>
<td>random number generator seed for the multistart solver</td>
<td>1019</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>NLP_MSW_SOLIDX</td>
<td>index of the multistart solution to be loaded</td>
<td>0</td>
</tr>
<tr>
<td>NLP_MSW_XKKTRAD_FACTOR</td>
<td>KKT solution neighborhood factor in multistart search</td>
<td>0.85</td>
</tr>
<tr>
<td>NLP_MSW_XNULRAD_FACTOR</td>
<td>initial solution neighborhood factor in multistart search</td>
<td>0.5</td>
</tr>
<tr>
<td>NLP_PRELEVEL</td>
<td>controls the amount and type of NLP pre-solving</td>
<td>126</td>
</tr>
<tr>
<td>NLP_PSTEP_FIN Ted DIFF</td>
<td>value of the step length in computing the derivatives using finite differences</td>
<td>5e-7</td>
</tr>
<tr>
<td>NLP_QUADCHK</td>
<td>flag for checking if NLP is quadratic</td>
<td>1</td>
</tr>
<tr>
<td>NLP_REDGTOL</td>
<td>tolerance for the gradients of nonlinear functions</td>
<td>1e-7</td>
</tr>
<tr>
<td>NLP_SOLVER</td>
<td>type of nonlinear solver</td>
<td>7</td>
</tr>
<tr>
<td>NLP_SOLVE_AS_LP</td>
<td>flag indicating if the nonlinear model will be solved as an LP</td>
<td>0</td>
</tr>
<tr>
<td>NLPSTALL_ITRLMT</td>
<td>iteration limit before a sequence of non-improving NLP iterations is declared as stalling</td>
<td>100</td>
</tr>
<tr>
<td>NLP_STARTPOINT</td>
<td>flag for using initial starting solution for NLP</td>
<td>1</td>
</tr>
<tr>
<td>NLP_SUBSOLVER</td>
<td>type of nonlinear subsolver</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USECUTOFFOBJ</td>
<td>flag to use parameter NLP_CUTOFFOBJ</td>
<td>0</td>
</tr>
<tr>
<td>NLP_USE_CRASH</td>
<td>flag for using simple crash routines for initial solution</td>
<td>0</td>
</tr>
<tr>
<td>NLP_USE_LINDO_CRASH</td>
<td>flag for using advanced crash routines for initial solution</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_SDP</td>
<td>flag to use SDP solver for POSD constraint</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_SELCON EVAL</td>
<td>flag for using selective constraint evaluations for solving NLP</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_SLP</td>
<td>flag for using sequential linear programming step directions for updating solution</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_STEEPEDGE</td>
<td>flag for using steepest edge directions for updating solution</td>
<td>0</td>
</tr>
</tbody>
</table>

5.24.6.6 Global Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOP_ABSOPTTOL</td>
<td>absolute optimality tolerance</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>GOP_ALGREFORMMD</td>
<td>algebraic reformulation rule for a GOP</td>
<td>18</td>
</tr>
<tr>
<td>GOP_BBSRCHMD</td>
<td>node selection rule in GOP branch-and-bound</td>
<td>1</td>
</tr>
<tr>
<td>GOP_BNDLIM</td>
<td>max magnitude of variable bounds used in GOP convexification</td>
<td>1e10</td>
</tr>
<tr>
<td>GOP_BOXTOL</td>
<td>minimal width of variable intervals</td>
<td>1e-6</td>
</tr>
<tr>
<td>GOP_BRANCHMD</td>
<td>direction to branch first when branching on a variable</td>
<td>5</td>
</tr>
<tr>
<td>GOP_BRANCH_LIMIT</td>
<td>limit on the total number of branches to be created in GOP tree</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_CMINLP</td>
<td>flag indicating if GOP exploits convex MINLP model</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>GOP_CONIC_REFORM</td>
<td>flag indicating if GOP explore conic reformulation</td>
<td>1</td>
</tr>
<tr>
<td>GOP_CORELEVEL</td>
<td>strategy of GOP branch-and-bound</td>
<td>14</td>
</tr>
<tr>
<td>GOP_DECOMPTMD</td>
<td>decomposition point selection rule in GOP branch-and-bound</td>
<td>1</td>
</tr>
<tr>
<td>GOP_DELTATOL</td>
<td>delta tolerance in GOP convexification</td>
<td>1e-7</td>
</tr>
<tr>
<td>GOP_FLTTOL</td>
<td>floating-point tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td>GOP_HEU_MODE</td>
<td>heuristic used in global solver</td>
<td>0</td>
</tr>
<tr>
<td>GOP_ITRLIM</td>
<td>GOP iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>GOP_ITRLIM_IPM</td>
<td>total barrier iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_ITRLIM_NLP</td>
<td>total nonlinear iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_ITRLIM_SIM</td>
<td>total simplex iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_LIM_MODE</td>
<td>flag indicating which heuristic limit on sub-solver in GOP is based</td>
<td>1</td>
</tr>
<tr>
<td>GOP_LINEARZ</td>
<td>flag indicating if GOP exploits linearizable model</td>
<td>1</td>
</tr>
<tr>
<td>GOP_LSOLBRANLIM</td>
<td>branch limit until finding a new nonlinear solution</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_MAXWIDMD</td>
<td>maximum width flag for the global solution</td>
<td>0</td>
</tr>
<tr>
<td>GOP_MULTILINEAR</td>
<td>flag indicating if GOP exploits multi linear feature</td>
<td>1</td>
</tr>
<tr>
<td>GOP_NUM_THREADS</td>
<td>number of parallel threads to be used when solving a nonlinear model with the global optimization solver</td>
<td>1</td>
</tr>
<tr>
<td>GOP_OBJ_THRESHOLD</td>
<td>threshold of objective value in the GOP solver</td>
<td>-1e+30</td>
</tr>
<tr>
<td>GOP_OPTCHKMD</td>
<td>criterion used to certify the global optimality</td>
<td>2</td>
</tr>
<tr>
<td>GOP_OPT_MODE</td>
<td>mode for GOP optimization</td>
<td>1</td>
</tr>
<tr>
<td>GOP_POSTLEVEL</td>
<td>amount and type of GOP postsolving</td>
<td>6</td>
</tr>
<tr>
<td>GOP_PRELEVEL</td>
<td>amount and type of GOP presolving</td>
<td>30</td>
</tr>
<tr>
<td>GOP_QUAD_METHOD</td>
<td>specifies if the GOP solver should solve the model as a QP when applicable</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_RELBRNDMD</td>
<td>reliable rounding in the GOP branch-and-bound</td>
<td>0</td>
</tr>
<tr>
<td>GOP_RELOPTTOL</td>
<td>relative optimality tolerance</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>GOP_SOLLIM</td>
<td>integer solution limit for GOP branch-and-bound</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_SUBOUT_MODE</td>
<td>substituting out fixed variables</td>
<td>1</td>
</tr>
<tr>
<td>GOP_TIMLIM</td>
<td>time limit in seconds for GOP branch-and-bound</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>GOP_USEBNDLIM</td>
<td>max magnitude of variable bounds flag for GOP convexification</td>
<td>2</td>
</tr>
<tr>
<td>GOP_WIDTOL</td>
<td>maximal width of variable intervals</td>
<td>1e-4</td>
</tr>
<tr>
<td>USEGOP</td>
<td>use global optimization</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CORE.ORDER_BY_STAGE</td>
<td>order nontemporal models or not</td>
<td>1</td>
</tr>
<tr>
<td>REPORTEVSOl</td>
<td>solve and report the expected value solution</td>
<td>0</td>
</tr>
<tr>
<td>SAMP.CDSINC</td>
<td>correlation matrix diagonal shift increment</td>
<td>1e-6</td>
</tr>
<tr>
<td>SAMP.NCM_CUTOBJ</td>
<td>objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem</td>
<td>1e-30</td>
</tr>
<tr>
<td>SAMP_NCM_DSTORAGE</td>
<td>flag to enable or disable sparse mode in NCM computations</td>
<td>-1</td>
</tr>
<tr>
<td>SAMP_NCM_ITERLIM</td>
<td>iteration limit for NCM method</td>
<td>100</td>
</tr>
<tr>
<td>SAMP_NCM_METHOD</td>
<td>bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem</td>
<td>5</td>
</tr>
<tr>
<td>SAMP_NCM_OPTTOL</td>
<td>optimality tolerance for NCM method</td>
<td>1e-7</td>
</tr>
<tr>
<td>SAMP_SCALE</td>
<td>flag to enable scaling of raw sample data</td>
<td>0</td>
</tr>
<tr>
<td>STOC_ABSOPTTOL</td>
<td>absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>STOC_ADD_MPI</td>
<td>flag to use add-instructions mode when building deteq</td>
<td>0</td>
</tr>
<tr>
<td>STOC_ALD_DUAL_FEASTOL</td>
<td>dual feasibility tolerance for ALD</td>
<td>1e-4</td>
</tr>
<tr>
<td>STOC_ALD_DUAL_STEPLEN</td>
<td>dual step length for ALD</td>
<td>0.9</td>
</tr>
<tr>
<td>STOC_ALD_INNER_ITER_LIM</td>
<td>inner loop iteration limit for ALD</td>
<td>1000</td>
</tr>
<tr>
<td>STOC_ALD_OUTER_ITER_LIM</td>
<td>outer loop iteration limit for ALD</td>
<td>200</td>
</tr>
<tr>
<td>STOC_ALD_PRIMAL_FEASTOL</td>
<td>primal feasibility tolerance for ALD</td>
<td>1e-4</td>
</tr>
<tr>
<td>STOC_ALD_PRIMAL_STEPLEN</td>
<td>primal step length for ALD</td>
<td>0.5</td>
</tr>
<tr>
<td>STOC_AUTOAGGR</td>
<td>flag to enable or disable autoaggregation</td>
<td>1</td>
</tr>
<tr>
<td>STOC_BENCHMARK_SCEN</td>
<td>benchmark scenario to compare EVPI and EVMU against</td>
<td>-2</td>
</tr>
<tr>
<td>STOC_BIGM</td>
<td>big-M value for linearization and penalty functions</td>
<td>1e7</td>
</tr>
<tr>
<td>STOC_BUCKET_SIZE</td>
<td>bucket size in Benders decomposition</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_CALC_EVPI</td>
<td>flag to enable or disable calculation of EVPI</td>
<td>1</td>
</tr>
<tr>
<td>STOC_CORRELATION_TYPE</td>
<td>correlation type associated with correlation matrix</td>
<td>0</td>
</tr>
<tr>
<td>STOC_DEQOPT</td>
<td>method to solve the DETEQ problem</td>
<td>0</td>
</tr>
<tr>
<td>STOC_DETEQ_TYPE</td>
<td>type of deterministic equivalent</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_DS_SUBFORM</td>
<td>subproblem formulation to use in DirectSearch</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_ELIM_FXVAR</td>
<td>flag to enable elimination of fixed variables from deteq MPI</td>
<td>1</td>
</tr>
<tr>
<td>STOC_INFBOUND</td>
<td>value to truncate infinite bounds at non-leaf nodes</td>
<td>1e9</td>
</tr>
<tr>
<td>STOC_ITER_LIM</td>
<td>iteration limit for stochastic solver</td>
<td>infinity</td>
</tr>
<tr>
<td>STOC_MAP_MPI2LP</td>
<td>flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>STOC_MAX_NUMSCENS</td>
<td>maximum number of scenarios before forcing automatic sampling</td>
<td>40000</td>
</tr>
<tr>
<td>STOC_METHOD</td>
<td>stochastic optimization method to solve the model</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NAMEDATA_LEVEL</td>
<td>name data level</td>
<td>1</td>
</tr>
<tr>
<td>STOC_NODELAP_PRELEVEL</td>
<td>presolve level solving node-models</td>
<td>0</td>
</tr>
<tr>
<td>STOC_NSAMPLE_PER_STAGE</td>
<td>list of sample sizes per stage (starting at stage 2)</td>
<td></td>
</tr>
<tr>
<td>STOC_NSAMPLE_SPAR</td>
<td>common sample size per stochastic parameter</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NSAMPLE_STAGE</td>
<td>common sample size per stage</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NUM_THREADS</td>
<td>number of parallel threads</td>
<td>1</td>
</tr>
<tr>
<td>STOC_RELOPTTOL</td>
<td>relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>STOC_REL_DSTEPTOL</td>
<td>dual-step tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>STOC_REL_PSTEPTOL</td>
<td>primal-step tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>STOC_REOPT</td>
<td>reoptimization method to solve the node-models</td>
<td>0</td>
</tr>
<tr>
<td>STOC_RG_SEED</td>
<td>seed to initialize the random number generator</td>
<td>1031</td>
</tr>
<tr>
<td>STOC_SAMP_CONT_ONLY</td>
<td>flag to restrict sampling to continuous stochastic parameters only or not</td>
<td>1</td>
</tr>
<tr>
<td>STOC_SBD_MAXCUTS</td>
<td>max cuts to generate for master problem</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_SBD_NUMCANDID</td>
<td>maximum number of candidate solutions to generate at SBD root</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_SBD_OBJCUTFLAG</td>
<td>flag to enable objective cut in SBD master problem</td>
<td>1</td>
</tr>
<tr>
<td>STOC_SBD_OBJCUTVAL</td>
<td>RHS value of objective cut in SBD master problem</td>
<td>1e-30</td>
</tr>
<tr>
<td>STOC_SHARE_BEGSTAGE</td>
<td>stage beyond which node-models are shared</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_TIME_LIM</td>
<td>time limit for stochastic solver</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>STOC_TOPOPT</td>
<td>optimization method to solve the root problem</td>
<td>0</td>
</tr>
<tr>
<td>STOC_VARCONTROL_METHOD</td>
<td>sampling method for variance reduction</td>
<td>1</td>
</tr>
<tr>
<td>STOC_WSBAS</td>
<td>warm start basis for wait-see model</td>
<td>-1</td>
</tr>
<tr>
<td>SVR_LS_ANTITHETIC</td>
<td>Sample variance reduction map to Lindo Antithetic algorithm</td>
<td></td>
</tr>
<tr>
<td>SVR_LS_LATINSQUARE</td>
<td>Sample variance reduction map to Lindo Latin Square algorithm</td>
<td></td>
</tr>
<tr>
<td>SVR_LS_MONTECARLO</td>
<td>Sample variance reduction map to Lindo Monte Carlo algorithm</td>
<td></td>
</tr>
</tbody>
</table>

### 5.24.6.8 IIS and IUS Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIS</td>
<td>run the IIS finder if the problem is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>IIS_ANALYZE_LEVEL</td>
<td>controls the level of analysis when locating an IIS</td>
<td>1</td>
</tr>
<tr>
<td>IIS_GETMODE</td>
<td>flag that controls whether variable bounds in the IIS should be retrieved or the integer restrictions</td>
<td>0</td>
</tr>
<tr>
<td>IIS_INFEAS_NORM</td>
<td>specifies the norm to measure infeasibilities in IIS search</td>
<td>0</td>
</tr>
<tr>
<td>IIS_ITER_LIMIT</td>
<td>the iteration limit for IIS search</td>
<td>-1</td>
</tr>
<tr>
<td>IIS_METHOD</td>
<td>specifies the method to use in analyzing infeasible models to locate an IIS</td>
<td>0</td>
</tr>
<tr>
<td>IIS_PRINT_LEVEL</td>
<td>specifies the amount of print to do during IIS search</td>
<td>2</td>
</tr>
<tr>
<td>IIS_REOPT</td>
<td>specifies which optimization method to use when starting from a given basis</td>
<td>0</td>
</tr>
<tr>
<td>IIS_TIME_LIMIT</td>
<td>the time limit for IIS search</td>
<td>-1</td>
</tr>
<tr>
<td>IIS_TOPOPT</td>
<td>specifies which optimization method to use when there is no previous basis</td>
<td>0</td>
</tr>
<tr>
<td>IIS_USE_EFILTER</td>
<td>flag that controls whether the Elastic Filter should be enabled as the supplementary filter in analyzing infeasible models</td>
<td>0</td>
</tr>
<tr>
<td>IIS_USE_GOP</td>
<td>flag that controls whether the global optimizer should be enabled in analyzing infeasible NLP models</td>
<td>0</td>
</tr>
<tr>
<td>IIS_USE_SFILTER</td>
<td>flag indicating is sensitivity filter will be used during IIS search</td>
<td>1</td>
</tr>
<tr>
<td>IUS</td>
<td>run the IUS finder if the problem is unbounded</td>
<td>0</td>
</tr>
<tr>
<td>IUS_ANALYZE_LEVEL</td>
<td>controls the level of analysis when locating an IUS</td>
<td>2</td>
</tr>
</tbody>
</table>

5.24.6.9 Link Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKRANGE</td>
<td>calculate feasible range for variables</td>
<td>range.gdx</td>
</tr>
<tr>
<td>READPARAMS</td>
<td>read Lindo parameter file</td>
<td></td>
</tr>
<tr>
<td>WRITEMPG</td>
<td>write deterministic equivalent in MPS format</td>
<td></td>
</tr>
<tr>
<td>WRITEMPS</td>
<td>write deterministic equivalent in MPS format</td>
<td></td>
</tr>
<tr>
<td>WRITEDEMPI</td>
<td>write (S)MPI file of processed model</td>
<td></td>
</tr>
<tr>
<td>WRITEDEMPS</td>
<td>write (S)MPI file of processed model</td>
<td></td>
</tr>
</tbody>
</table>

5.24.7 Detailed Descriptions of GAMS/Lindo Options

**CHECKRANGE** *(string)*: calculate feasible range for variables

If this option is set, Lindo calculates the feasible range (determined by an upper and lower bound) for every variable in each equation while all other variables are fixed to their level. If set, the value of this option defines the name of the GDX file where the results are written to.

For every combination of equation- and variable block there will be one symbol in the format *EquBlock_VarBlock(equ_Ind_1, ..., equ_Ind_M, var_Ind_1, ..., var_Ind_N, directions)*.

Default: range.gdx

**CORE_ORDER_BY_STAGE** *(integer)*: order nontemporal models or not
Order non-temporal models or not.

Default: 1

**DECOMPOSITION_TYPE** (integer): decomposition to be performed on a linear or mixed integer model

This refers to the type of decomposition to be performed on a linear or mixed integer model.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides which type of decomposition to use</td>
</tr>
<tr>
<td>1</td>
<td>Solver does not perform any decompositions and uses the original model</td>
</tr>
<tr>
<td>2</td>
<td>Attempt total decomposition</td>
</tr>
<tr>
<td>3</td>
<td>Decomposed model will have dual angular structure</td>
</tr>
<tr>
<td>4</td>
<td>Decomposed model will have block angular structure</td>
</tr>
<tr>
<td>5</td>
<td>Decomposed model will have both dual and block angular structure</td>
</tr>
</tbody>
</table>

**FIND_BLOCK** (integer): graph partitioning method to find block structures

Specifies the graph partitioning method to find block structures.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use an edge-weight minimizing graph partitioning heuristic</td>
</tr>
<tr>
<td>1</td>
<td>Use a vertex-weight minimizing graph partitioning heuristic</td>
</tr>
</tbody>
</table>

**FIND_SYMMETRY_LEVEL** (integer): specifies the symmetry finding level.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Finding orbit only without MIP preprocessing</td>
</tr>
<tr>
<td>1</td>
<td>Finding orbit only with MIP preprocessing</td>
</tr>
<tr>
<td>2</td>
<td>Finding generators without MIP preprocessing</td>
</tr>
<tr>
<td>3</td>
<td>Finding generators with MIP preprocessing</td>
</tr>
<tr>
<td>4</td>
<td>Finding the first generator without MIP preprocessing</td>
</tr>
<tr>
<td>5</td>
<td>Finding the first generator with MIP preprocessing</td>
</tr>
</tbody>
</table>

**FIND_SYMMETRY_PRINT_LEVEL** (integer): specifies print level for symmetry finding

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nothing printed</td>
</tr>
<tr>
<td>+2</td>
<td>General information</td>
</tr>
<tr>
<td>+4</td>
<td>Time information</td>
</tr>
<tr>
<td>+8</td>
<td>Orbit information</td>
</tr>
<tr>
<td>+16</td>
<td>Partition information</td>
</tr>
</tbody>
</table>
**GOP_ABSOPTTOL** *(real)*: absolute optimality tolerance

Synonym: ABSOPTTOL

This value is the GOP absolute optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: GAMS OptCA

**GOP_ALGREFORMMD** *(integer)*: algebraic reformulation rule for a GOP

Synonym: ALGREFORMMD

This controls the algebraic reformulation rule for a GOP. The algebraic reformulation and analysis is very crucial in building a tight convex envelope to enclose the nonlinear/nonconvex functions. A lower degree of overestimation on convex envelopes helps increase the convergence rate to the global optimum.

Default: 18

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Rearrange and collect terms</td>
</tr>
<tr>
<td>+4</td>
<td>Expand all parentheses</td>
</tr>
<tr>
<td>+8</td>
<td>Retain nonlinear functions</td>
</tr>
<tr>
<td>+16</td>
<td>Selectively expand parentheses</td>
</tr>
</tbody>
</table>

**GOP_BBSRCHMD** *(integer)*: node selection rule in GOP branch-and-bound

Synonym: BBSRCHMD

This specifies the node selection rule for choosing between all active nodes in the GOP branch-and-bound tree when solving global optimization programs.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Depth first search</td>
</tr>
<tr>
<td>1</td>
<td>Choose node with worst bound</td>
</tr>
</tbody>
</table>

**GOP_BNDLIM** *(real)*: max magnitude of variable bounds used in GOP convexification

Synonym: BNDLIM

This value specifies the maximum magnitude of variable bounds used in the GOP convexification. Any lower bound smaller than the negative of this value will be treated as the negative of this value. Any upper bound greater than this value will be treated as this value. This helps the global solver focus on more productive domains.

Default: 1e10

**GOP_BOXTOL** *(real)*: minimal width of variable intervals

Synonym: BOXTOL

This value specifies the minimal width of variable intervals in a box allowed to branch.

Default: 1e-6
GOP_BRANCHMD (integer): direction to branch first when branching on a variable

Synonym: BRANCHMD

This specifies the direction to branch first when branching on a variable. The branch variable is selected as the one that holds the largest magnitude in the measure.

Default: 5

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Absolute width</td>
</tr>
<tr>
<td>1</td>
<td>Locally relative width</td>
</tr>
<tr>
<td>2</td>
<td>Globally relative width</td>
</tr>
<tr>
<td>3</td>
<td>Globally relative distance from the convex minimum to the bounds</td>
</tr>
<tr>
<td>4</td>
<td>Absolute violation between the function and its convex envelope at the convex minimum</td>
</tr>
<tr>
<td>5</td>
<td>Relative violation between the function and its convex envelope at the convex minimum</td>
</tr>
</tbody>
</table>

GOP_BRANCH_LIMIT (integer): limit on the total number of branches to be created in GOP tree

Synonym: BRANCH_LIMIT

This is the limit on the total number of branches to be created during branch-and-bound in GOP tree. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible solution was found, it will be installed as the incumbent (best known) solution.

Range: \{-1, ..., \infty\}

Default: -1

GOP_CMINLP (integer): flag indicating if GOP exploits convex MINLP model

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

GOP_CONIC_REFORM (integer): flag indicating if GOP explore conic reformulation

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

GOP_CORELEVEL (integer): strategy of GOP branch-and-bound

Synonym: CORELEVEL

This controls the strategy of GOP branch-and-bound procedure.
Default: 14
### GOP_DECOMPPTMD

**Type:** (integer)

**Description:** decomposition point selection rule in GOP branch-and-bound

**Synonym:** DECOMPPTMD

This specifies the decomposition point selection rule. In the branch step of GOP branch-and-bound, a branch point M is selected to decompose the selected variable interval [Lb, Ub] into two subintervals, [Lb, M] and [M, Ub].

**Default:** 1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>LP convex relaxation</td>
</tr>
<tr>
<td>+4</td>
<td>NLP solving</td>
</tr>
<tr>
<td>+8</td>
<td>Box Branching</td>
</tr>
</tbody>
</table>

### GOP_DELTATOL

**Type:** (real)

**Description:** delta tolerance in GOP convexification

**Synonym:** DELTATOL

This value is the delta tolerance in the GOP convexification. It is a measure of how closely the additional constraints added as part of convexification should be satisfied.

**Default:** 1e-7

### GOP_FLTTOL

**Type:** (real)

**Description:** floating-point tolerance

**Synonym:** FLTTOL

This value is the GOP floating-point tolerance. It specifies the maximum rounding errors in the floating-point computation.

**Default:** 1e-10

### GOP_HEU_MODE

**Type:** (integer)

**Description:** heuristic used in global solver

**Synonym:** HEU_MODE

This specifies the heuristic used in the global solver to find a good solution. Typically, if a heuristic is used, this will put more efforts in searching for good solutions, and less in bound tightening.

**Default:** 0

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No heuristic is used</td>
</tr>
<tr>
<td>1</td>
<td>A simple heuristic is used</td>
</tr>
</tbody>
</table>

### GOP_ITRLIM

**Type:** (real)

**Description:** GOP iteration limit
Synonym: ITRLIM

This is the total iteration limit (including simplex, barrier and nonlinear iteration) summed over branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: \([-1, \infty]\]
Default: infinity

**GOP_ITRLIM_JPM** (*real*): total barrier iteration limit summed over all branches in GOP

Synonym: ITRLIM_JPM

This is the total barrier iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: \([-1, \infty]\]
Default: -1

**GOP_ITRLIM_NLP** (*real*): total nonlinear iteration limit summed over all branches in GOP

Synonym: ITRLIM_NLP

This is the total nonlinear iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: \([-1, \infty]\]
Default: -1

**GOP_ITRLIM_SIM** (*real*): total simplex iteration limit summed over all branches in GOP

Synonym: ITRLIM_SIM

This is the total simplex iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: \([-1, \infty]\]
Default: -1

**GOP_LIM_MODE** (*integer*): flag indicating which heuristic limit on sub-solver in GOP is based

Synonym: LIM_MODE

This is a flag indicating which heuristic limit on sub-solver in GOP is based.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No limit</td>
</tr>
<tr>
<td>1</td>
<td>Time based limit</td>
</tr>
<tr>
<td>2</td>
<td>Iteration based limit</td>
</tr>
<tr>
<td>3</td>
<td>Both time and iteration based limit</td>
</tr>
</tbody>
</table>
**GOP_LINEARZ** *(integer)*: flag indicating if GOP exploits linearizable model

This is a flag indicating if GOP exploits linearizable model.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Exploit linearizable model</td>
</tr>
<tr>
<td>1</td>
<td>Do not exploit linearizable model</td>
</tr>
</tbody>
</table>

**GOP_LSOLBRANLIM** *(integer)*: branch limit until finding a new nonlinear solution

Synonym: LSOLBRANLIM

This value controls the branch limit until finding a new nonlinear solution since the last nonlinear solution is found. The default value is -1, which means no branch limit is imposed.

Range: \{-1, ..., \infty\}

Default: -1

**GOP_MAXWIDMD** *(integer)*: maximum width flag for the global solution

Synonym: MAXWIDMD

This is the maximum width flag for the global solution. The GOP branch-and-bound may continue contracting a box with an incumbent solution until its maximum width is smaller than **GOP_WIDTOL**.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The maximum width criterion is suppressed</td>
</tr>
<tr>
<td>1</td>
<td>The maximum width criterion is performed</td>
</tr>
</tbody>
</table>

**GOP_MULTILINEAR** *(integer)*: flag indicating if GOP exploits multi linear feature

This is a flag indicating if GOP exploits multi linear feature.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

**GOP_NUM_THREADS** *(integer)*: number of parallel threads to be used when solving a nonlinear model with the global optimization solver

This value specifies the number of parallel threads to be used when solving a nonlinear model with the global optimization solver.

Default: 1
**GOP_OBJ_THRESHOLD** (*real*): threshold of objective value in the GOP solver

This value specifies the threshold of objective value in the GOP solver. For min problem, if current incumbent solution is less than the threshold GOP solver will stop.

Range: $[-1e+30, \infty]$  
Default: $-1e+30$

**GOP_OPTCHKMD** (*integer*): criterion used to certify the global optimality

Synonym: OPTCHKMD

This specifies the criterion used to certify the global optimality. When this value is 0, the absolute deviation of objective lower and upper bounds should be smaller than **GOP_ABSOPTTOL** at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than **GOP_RELOPTTOL** at the global optimum. 2 means either absolute or relative tolerance is satisfied at global optimum.

Default: 2

**GOP_OPT_MODE** (*integer*): mode for GOP optimization

Synonym: OPT_MODE

This specifies the mode for GOP optimization.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Global search for a feasible solution (thus a feasibility certificate)</td>
</tr>
<tr>
<td>1</td>
<td>Global search for an optimal solution</td>
</tr>
<tr>
<td>2</td>
<td>Global search for an unboundedness certificate</td>
</tr>
</tbody>
</table>

**GOP_POSTLEVEL** (*integer*): amount and type of GOP postsolving

Synonym: POSTLEVEL

This controls the amount and type of GOP post-solving. The default value is: $6 = 2+4$ meaning to do both of the below options.

Default: 6

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Apply LSgetBestBound</td>
</tr>
<tr>
<td>+4</td>
<td>Reoptimize variable bounds</td>
</tr>
</tbody>
</table>

**GOP_PRELEVEL** (*integer*): amount and type of GOP presolving

Synonym: PRELEVEL

This controls the amount and type of GOP pre-solving. The default value is: $30 = 2+4+8+16$ meaning to do all of the below options.

Default: 30
GOP QUAD_METHOD (integer): specifies if the GOP solver should solve the model as a QP when applicable

Default: −1

GOP RELBRNDMD (integer): reliable rounding in the GOP branch-and-bound

Synonym: RELBRNDMD

This controls the reliable rounding rule in the GOP branch-and-bound. The global solver applies many suboptimizations to estimate the lower and upper bounds on the global optimum. A rounding error or numerical instability could unintentionally cut off a good solution. A variety of reliable approaches are available to improve the precision.

Default: 0

GOP RELOPTTOL (real): relative optimality tolerance

Synonyms: OPTTOL RELOPTTOL

This value is the GOP relative optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: GAMS OptCR

GOP SOLLIM (integer): integer solution limit for GOP branch-and-bound

Range: \{-1, ..., ∞\}

Default: -1

GOP SUBOUT MODE (integer): substituting out fixed variables

Synonym: SUBOUT_MODE

This is a flag indicating whether fixed variables are substituted out of the instruction list used in the global solver.

Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not substitute out fixed variables</td>
</tr>
<tr>
<td>1</td>
<td>Substitute out fixed variables</td>
</tr>
</tbody>
</table>

**GOP_TIMLIM** (*integer*): time limit in seconds for GOP branch-and-bound

Synonym: TIMLIM

This is the time limit in seconds for GOP branch-and-bound.

Range: \{-1, ..., \infty\}

Default: GAMS ResLim

**GOP_USEBNDLIM** (*integer*): max magnitude of variable bounds flag for GOP convexification

Synonym: USEBNDLIM

This value is a flag for the parameter **GOP_BNDLIM**.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use the bound limit on the variables</td>
</tr>
<tr>
<td>1</td>
<td>Use the bound limit right at the beginning of global optimization</td>
</tr>
<tr>
<td>2</td>
<td>Use the bound limit after the initial local optimization if selected</td>
</tr>
</tbody>
</table>

**GOP_WIDTOL** (*real*): maximal width of variable intervals

Synonym: WIDTOL

This value specifies the maximal width of variable intervals for a box to be considered as an incumbent box containing an incumbent solution. It is used when **GOP_MAXWIDMD** is set at 1.

Default: $1e^{-4}$

**IIS** (*boolean*): run the IIS finder if the problem is infeasible

Default: 0

**IIS_ANALYZE_LEVEL** (*integer*): controls the level of analysis when locating an IIS

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Search for necessary rows</td>
</tr>
<tr>
<td>+2</td>
<td>Search for necessary columns</td>
</tr>
<tr>
<td>+4</td>
<td>Search for sufficient rows</td>
</tr>
<tr>
<td>+8</td>
<td>Search for sufficient columns</td>
</tr>
<tr>
<td>+16</td>
<td>Consider integrality restrictions as the potential cause of infeasibility and include it in the analysis</td>
</tr>
<tr>
<td>+32</td>
<td>Compute the underlying LTF matrix and use this as the basis of a ranking score to guide the IIS run</td>
</tr>
<tr>
<td>+64</td>
<td>If the underlying matrix is totally decomposable, rank blocks w.r.t. their sizes and debug the smallest independent infeasible block</td>
</tr>
<tr>
<td>+128</td>
<td>Use the nonzero structure of the underlying matrix to compute a ranking score to guide the IIS run</td>
</tr>
<tr>
<td>+256</td>
<td>Treat iter/time limits as intractability</td>
</tr>
</tbody>
</table>
**IIS_GETMODE** *(integer)*: flag that controls whether variable bounds in the IIS should be retrieved or the integer restrictions

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Variable bound</td>
</tr>
<tr>
<td>1</td>
<td>Integer restrictions</td>
</tr>
</tbody>
</table>

**IIS_INFEAS_NORM** *(integer)*: specifies the norm to measure infeasibilities in IIS search

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use L-1 norm</td>
</tr>
<tr>
<td>2</td>
<td>L-infinity norm</td>
</tr>
</tbody>
</table>

**IIS_ITER_LIMIT** *(integer)*: the iteration limit for IIS search

Range: \{-1, ..., \infty\}

Default: -1

**IIS_METHOD** *(integer)*: specifies the method to use in analyzing infeasible models to locate an IIS

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default filter</td>
</tr>
<tr>
<td>1</td>
<td>Standard deletion filter</td>
</tr>
<tr>
<td>2</td>
<td>Standard additive filter</td>
</tr>
<tr>
<td>3</td>
<td>Generalized-binary-search filter</td>
</tr>
<tr>
<td>4</td>
<td>Depth-first-binary-search filter</td>
</tr>
<tr>
<td>5</td>
<td>Fast-scan filter</td>
</tr>
<tr>
<td>6</td>
<td>Standard elastic filter</td>
</tr>
</tbody>
</table>

**IIS_PRINT_LEVEL** *(integer)*: specifies the amount of print to do during IIS search

Default: 2

**IIS_REOPT** *(integer)*: specifies which optimization method to use when starting from a given basis

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Free</td>
</tr>
<tr>
<td>1</td>
<td>Primal Simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual Simplex</td>
</tr>
<tr>
<td>3</td>
<td>Barrier</td>
</tr>
<tr>
<td>4</td>
<td>NLP</td>
</tr>
</tbody>
</table>
**IIS_TIME_LIMIT** *(integer)*: the time limit for IIS search

Range: \{-1, ..., \infty\}

Default: -1

**IIS_TOPOPT** *(integer)*: specifies which optimization method to use when there is no previous basis

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Free</td>
</tr>
<tr>
<td>1</td>
<td>Primal Simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual Simplex</td>
</tr>
<tr>
<td>3</td>
<td>Barrier</td>
</tr>
<tr>
<td>4</td>
<td>NLP</td>
</tr>
</tbody>
</table>

**IIS_USE_EFILTER** *(integer)*: flag that controls whether the Elastic Filter should be enabled as the supplementary filter in analyzing infeasible models

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Do not use elastic filter</td>
</tr>
<tr>
<td>1</td>
<td>Use elastic filter</td>
</tr>
</tbody>
</table>

**IIS_USE_GOP** *(integer)*: flag that controls whether the global optimizer should be enabled in analyzing infeasible NLP models

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Do not use GOP</td>
</tr>
<tr>
<td>1</td>
<td>Use GOP</td>
</tr>
</tbody>
</table>

**IIS_USE_SFILTER** *(integer)*: flag indicating is sensitivity filter will be used during IIS search

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Do not use sensitivity filter</td>
</tr>
<tr>
<td>1</td>
<td>Use sensitivity filter</td>
</tr>
</tbody>
</table>

**INSTRUCT_SUBOUT** *(integer)*: flag to specify how to deal with fixed variables in the instruction list

This is a flag indicating whether 1) fixed variables are substituted out of the instruction list, 2) performing numerical calculation on constant numbers and replacing with the results.
Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Substitutions will not be performed</td>
</tr>
<tr>
<td>1</td>
<td>Substitutions will be performed</td>
</tr>
</tbody>
</table>

**IPM**\_**BASIS**\_**REL**\_**TOL**\_**S** (*real*): maximum relative dual bound violation allowed in an optimal basic solution

Maximum relative dual bound violation allowed in an optimal basic solution.

Default: $1e^{-12}$

**IPM**\_**BASIS**\_**TOL**\_**S** (*real*): maximum absolute dual bound violation in an optimal basic solution

Maximum absolute dual bound violation in an optimal basic solution.

Default: $1e^{-7}$

**IPM**\_**BASIS**\_**TOL**\_**X** (*real*): maximum absolute primal bound violation allowed in an optimal basic solution

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default: $1e^{-7}$

**IPM**\_**BI**\_**LU**\_**TOL**\_**REL**\_**PIV** (*real*): relative pivot tolerance used in the LU factorization in the basis identification procedure

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

Range: \([0, 0.999999]\]

Default: $1e^{-2}$

**IPM**\_**CHECK**\_**CONVEXITY** (*integer*): flag to check convexity of a quadratic program using barrier solver

This is a flag to check convexity of a quadratic program using barrier solver.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Check convexity only without solving the model</td>
</tr>
<tr>
<td>0</td>
<td>Use barrier solver to check convexity</td>
</tr>
<tr>
<td>1</td>
<td>Do not use barrier solver to check convexity</td>
</tr>
</tbody>
</table>

**IPM**\_**CO**\_**TOL**\_**INFEAS** (*real*): maximum bound infeasibility tolerance for Conic solver

Maximum bound infeasibility tolerance for Conic solver.

Default: $1e^{-10}$

**IPM**\_**MAX**\_**ITERATIONS** (*integer*): ipm iteration limit
Controls the maximum number of iterations allowed in the interior-point optimizer.

Default: 1000

**IPM_NUM_THREADS** *(integer)*: number of threads to run the interior-point optimizer on

Number of threads to run the interior-point optimizer on. This value should be less than or equal to the actual number of processors or cores on a multi-core system.

Default: 1

**IPM_OFF_COL_TRH** *(integer)*: extent for detecting the offending columns in the Jacobian of the constraint matrix

Controls the extent for detecting the offending columns in the Jacobian of the constraint matrix. 0 means no offending columns will be detected. 1 means offending columns will be detected. In general, increasing the parameter value beyond the default value of 40 does not improve the result.

Default: 40

**IPM_TOL_DFEAS** *(real)*: dual feasibility tolerance

Dual feasibility tolerance used for linear and quadratic optimization problems.

Default: 1e-8

**IPM_TOL_DSAFE** *(real)*: controls the initial dual starting point

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the dual variables associated with constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [1e-4, $\infty$]

Default: 1

**IPM_TOL_INFEAS** *(real)*: infeasibility tolerance

This is the tolerance to declare the model primal or dual infeasible using the interior-point optimizer. A smaller number means the optimizer gets more conservative about declaring the model infeasible.

Default: 1e-10

**IPM_TOL_MU_RED** *(real)*: relative complementarity gap tolerance

Relative complementarity gap tolerance.

Default: 1e-16

**IPM_TOL_PATH** *(real)*: how close to follow the central path

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. For numerically unstable problems it might help to increase this parameter.

Range: [0, 0.5]

Default: 1e-8
IPM_TOL_PFEAS *(real)*: primal feasibility tolerance →

Primal feasibility tolerance used for linear and quadratic optimization problems.

Default: $1e^{-8}$

IPM_TOL_PSAFE *(real)*: controls the initial primal starting point →

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: $[1e^{-2}, \infty]$  
Default: 1

IPM_TOL_REL_STEP *(real)*: relative step size to the boundary →

Relative step size to the boundary for linear and quadratic optimization problems.

Range: $[0, 0.999999]$  
Default: 0.9999

IUS *(boolean)*: run the IUS finder if the problem is unbounded →

Default: 0

IUS_ANALYZE_LEVEL *(integer)*: controls the level of analysis when locating an IUS →

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Search for necessary columns</td>
</tr>
<tr>
<td>+8</td>
<td>Search for sufficient columns</td>
</tr>
</tbody>
</table>

LP_AIJ_ZEROTOL *(real)*: coefficient matrix zero tolerance →

Default: $2.22045e^{-16}$

LP_BIGM *(real)*: big-M for phase-I →

Default: $1e6$

LP_BNDINF *(real)*: big-M to truncate lower and upper bounds in single phase dual-simplex →

Default: $1e15$

LP_DPSWITCH *(integer)*: specifies whether LP primal-dual simplex switch is enabled or not →

Range: $\{0, ..., 1\}$  
Default: 1

LP_DRATIO *(integer)*: controls the dual min-ratio strategy →
Range: \{0, ..., 2\}

Default: 1

**LP\_DYNOBJFACT** *(real)*: Dynamic obj factor

Range: \([0, 1]\)

Default: 0.75

**LP\_DYNOBJMODE** *(integer)*: Dynamic obj mode

Default: 0

**LP\_ITRLMT** *(integer)*: simplex iteration limit

Synonym: SPLEX\_ITRLMT

Range: \([-1, ..., \infty]\)

Default: infinity

**LP\_PIV\_BIGTOL** *(real)*: simplex maximum pivot tolerance

Default: 1e-5

**LP\_PIV\_ZEROTOL** *(real)*: simplex pivot zero tolerance

Default: 1e-8

**LP\_PPARTIAL** *(integer)*: primal simplex partial pricing method

Range: \{0, ..., 3\}

Default: 0

**LP\_PRELEVEL** *(integer)*: controls the amount and type of LP pre-solving

This controls the amount and type of LP pre-solving to be used.

Default: 126

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Simple pre-solving</td>
</tr>
<tr>
<td>+4</td>
<td>Probing</td>
</tr>
<tr>
<td>+8</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+16</td>
<td>Elimination</td>
</tr>
<tr>
<td>+32</td>
<td>Dual reductions</td>
</tr>
<tr>
<td>+64</td>
<td>Use dual information</td>
</tr>
<tr>
<td>+512</td>
<td>Maximum pass</td>
</tr>
</tbody>
</table>

**LP\_RATRANGE** *(integer)*: controls the number of pivot-candidates to consider when searching for a stable pivot in LU decomposition

Range: \{1, ..., \infty\}
Default: 4

**LP_SCALE** *(integer):* scaling flag  
Synonym: SPLEX_SCALE

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Scaling is suppressed</td>
</tr>
<tr>
<td>1</td>
<td>Scaling is performed</td>
</tr>
</tbody>
</table>

**LP_SPRINT_COLFACT** *(integer):* maximum number of columns in Sprint as a factor of number of rows  
Range: \{1, ..., \infty\}

Default: 10

**LP_SPRINT_MAXPASS** *(integer):* maximum number of passes in Sprint method  
Range: \{1, ..., \infty\}

Default: 100

**LP_SPRINT_SUB** *(integer):* LP method for subproblem in Sprint method  
Default: 0

**MIP_ASCUTTOL** *(real):* MIP absolute cut tolerance  
This is the MIP absolute cut tolerance. If the value is less than or equal to zero, it will use the internal decided tolerace, otherwise it will use this value as the absolute tolerance for adding cuts.

Range: [-1.0, \infty]

Default: -1.0

**MIP_ASOPTTOL** *(real):* MIP absolute optimality tolerance  
This is the MIP absolute optimality tolerance. Solutions must beat the incumbent by at least this absolute amount to become the new, best solution.

Default: GAMS OptCA

**MIP_ADDCUTOBJTOL** *(real):* required objective improvement to continue generating cuts  
This specifies the minimum required improvement in the objective function for the cut generation phase to continue generating cuts.

Default: 1.5625e-5

**MIP_ADDCUTPER** *(real):* percentage of constraint cuts that can be added  

This determines how many constraint cuts can be added as a percentage of the number of original rows in an integer programming model.

Default: 0.75

**MIP_ADDCUTPER_TREE** *(real)*: percentage of constraint cuts that can be added at child nodes

This determines how many constraint cuts can be added at child nodes as a percentage of the number of original rows in an integer programming model.

Default: 0.5

**MIP_AGGCUTLIM_TOP** *(integer)*: max number of constraints involved in derivation of aggregation cut at root node

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the root node. The default is .1, which means that the solver will decide.

Range: \(-1, ..., \infty\)

Default: -1

**MIP_AGGCUTLIM_TREE** *(integer)*: max number of constraints involved in derivation of aggregation cut at tree nodes

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the tree nodes. The default is .1, which means that the solver will decide.

Range: \(-1, ..., \infty\)

Default: -1

**MIP_ANODES_SWITCH_DF** *(integer)*: threshold on active nodes for switching to depth-first search

This specifies the threshold on active nodes for switching to depth-first search rule.

Default: 50000

**MIP_AOPTTIMLIM** *(integer)*: time in seconds beyond which the relative optimality tolerance will be applied

This is the time in seconds beyond which the relative optimality tolerance, **MIP_PEROPTTOL** will be applied.

Default: 100

**MIP_BIGM_FOR_INTTOL** *(real)*: threshold for which coefficient of a binary variable would be considered as big-M

This value specifies the threshold for which the coefficient of a binary variable would be considered as big-M (when applicable).

Default: \(1e8\)

**MIP_BRANCHDIR** *(integer)*: first branching direction

This specifies the direction to branch first when branching on a variable.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Always branch up first</td>
</tr>
<tr>
<td>2</td>
<td>Always branch down first</td>
</tr>
</tbody>
</table>

**MIP_BRANCHRULE** *(integer)*: rule for choosing the variable to branch

This specifies the rule for choosing the variable to branch on at the selected node.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Basis rounding with pseudo reduced costs</td>
</tr>
<tr>
<td>2</td>
<td>Maximum infeasibility</td>
</tr>
<tr>
<td>3</td>
<td>Pseudo reduced costs only</td>
</tr>
<tr>
<td>4</td>
<td>Maximum coefficient only</td>
</tr>
<tr>
<td>5</td>
<td>Previous branching only</td>
</tr>
</tbody>
</table>

**MIP_BRANCH_LIMIT** *(integer)*: limit on the total number of branches to be created during branch and bound

This is the limit on the total number of branches to be created during branch-and-bound. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: \{-1, ..., \infty\}

Default: -1

**MIP_BRANCH_Prio** *(integer)*: controls how variable selection priorities are set and used

This controls how variable selection priorities are set and used.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>If the user has specified priorities then use them Otherwise let LINDO API decide</td>
</tr>
<tr>
<td>1</td>
<td>If user has specified priorities then use them Overwrite users choices if necessary</td>
</tr>
<tr>
<td>2</td>
<td>If user has specified priorities then use them Otherwise do not use any priorities</td>
</tr>
<tr>
<td>3</td>
<td>Let LINDO API set the priorities and ignore any user specified priorities</td>
</tr>
<tr>
<td>4</td>
<td>Binaries always have higher priority over general integers</td>
</tr>
</tbody>
</table>

**MIP_CUTDEPTH** *(integer)*: threshold value for the depth of nodes in the branch and bound tree

This controls a threshold value for the depth of nodes in the B&B tree, so cut generation will be less likely at those nodes deeper than this threshold.

Default: 8

**MIP_CUTFREQ** *(integer)*: frequency of invoking cut generation at child nodes
This controls the frequency of invoking cut generation at child nodes. The default value is 10, indicating that the MIP solver will try to generate cuts at every 10 nodes.

Default: 10

**MIP.CUTLEVEL.TOP** *(integer)*: combination of cut types to try at the root node when solving a MIP

This controls the combination of cut types to try at the root node when solving a MIP. Bit settings are used to enable the various cut types.

Default: 57342

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>GUB cover</td>
</tr>
<tr>
<td>+4</td>
<td>Flow cover</td>
</tr>
<tr>
<td>+8</td>
<td>Lifting</td>
</tr>
<tr>
<td>+16</td>
<td>Plant location</td>
</tr>
<tr>
<td>+32</td>
<td>Disaggregation</td>
</tr>
<tr>
<td>+64</td>
<td>Knapsack cover</td>
</tr>
<tr>
<td>+128</td>
<td>Lattice</td>
</tr>
<tr>
<td>+256</td>
<td>Gomory</td>
</tr>
<tr>
<td>+512</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+1024</td>
<td>GCD</td>
</tr>
<tr>
<td>+2048</td>
<td>Obj integrality</td>
</tr>
<tr>
<td>+4096</td>
<td>Basis Cuts</td>
</tr>
<tr>
<td>+8192</td>
<td>Cardinality Cuts</td>
</tr>
<tr>
<td>+16384</td>
<td>Disjunk Cuts</td>
</tr>
<tr>
<td>+32768</td>
<td>Soft Knapsack Cuts</td>
</tr>
</tbody>
</table>

**MIP.CUTLEVEL.TREE** *(integer)*: combination of cut types to try at child nodes in the branch and bound tree when solving a MIP

This controls the combination of cut types to try at child nodes in the B&B tree when solving a MIP.

Default: 53246

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>GUB cover</td>
</tr>
<tr>
<td>+4</td>
<td>Flow cover</td>
</tr>
<tr>
<td>+8</td>
<td>Lifting</td>
</tr>
<tr>
<td>+16</td>
<td>Plant location</td>
</tr>
<tr>
<td>+32</td>
<td>Disaggregation</td>
</tr>
<tr>
<td>+64</td>
<td>Knapsack cover</td>
</tr>
<tr>
<td>+128</td>
<td>Lattice</td>
</tr>
<tr>
<td>+256</td>
<td>Gomory</td>
</tr>
<tr>
<td>+512</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+1024</td>
<td>GCD</td>
</tr>
<tr>
<td>+2048</td>
<td>Obj integrality</td>
</tr>
<tr>
<td>+4096</td>
<td>Basis Cuts</td>
</tr>
<tr>
<td>+8192</td>
<td>Cardinality Cuts</td>
</tr>
<tr>
<td>+16384</td>
<td>Disjunk Cuts</td>
</tr>
<tr>
<td>+32768</td>
<td>Soft Knapsack Cuts</td>
</tr>
</tbody>
</table>
**MIP_CUTOFFOBJ** *(real)*: defines limit for branch and bound

If this is specified, then any part of the branch-and-bound tree that has a bound worse than this value will not be considered. This can be used to reduce the running time if a good bound is known.

Default: \(1e30\)

**MIP_CUTTIMLIM** *(integer)*: time to be spent in cut generation

This controls the total time to be spent in cut generation throughout the solution of a MIP. The default value is -1, indicating that no time limits will be imposed when generating cuts.

Range: \([-1, ..., \infty]\)

Default: -1

**MIP_DELTA** *(real)*: near-zero value used in linearizing nonlinear expressions

This refers to a near-zero value used in linearizing nonlinear expressions.

Default: \(1e-6\)

**MIP_DUAL_SOLUTION** *(integer)*: flag for computing dual solution of LP relaxation

This flag controls whether the dual solution to the LP relaxation that yielded the optimal MIP solution will be computed or not.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not calculate dual solution for LP relaxation</td>
</tr>
<tr>
<td>1</td>
<td>Calculate dual solution for LP relaxation</td>
</tr>
</tbody>
</table>

**MIP_FP_ITRLIM** *(integer)*: iteration limit for feasibility pump heuristic

This is the iteration limit in seconds for feasibility pump heuristic. A value of -1 means no iteration limit is imposed.

Default: 500

**MIP_FP_MODE** *(integer)*: mode for the feasibility pump heuristic

Controls the mode for the feasibility pump heuristic.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On until the first solution</td>
</tr>
<tr>
<td>2</td>
<td>Try to get more than one solution</td>
</tr>
</tbody>
</table>

**MIP_FP_OPT_METHOD** *(integer)*: optimization and reoptimization method for feasibility pump
heuristic←

This specifies optimization and reoptimization method for feasibility pump heuristic.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Primal simplex</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Barrier</td>
</tr>
</tbody>
</table>

MIP_FP_TIMLIM (real): time limit for feasibility pump heuristic←

This is the time limit in seconds for feasibility pump heuristic. A value of -1 implies no time limit is imposed.

Default: 1800

MIP_FP_WEIGHT (real): weight of the objective function in the feasibility pump←

Controls the weight of the objective function in the feasibility pump.

Range: [0, 1]

Default: 1

MIP_GENERAL_MODE (integer): general strategy in solving MIPs←

This value specifies the general strategy in solving MIPs.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>+2</td>
<td>Disable all time-driven events for reproducibility of runs</td>
</tr>
<tr>
<td>+16</td>
<td>Disable cut generation before branching</td>
</tr>
</tbody>
</table>

MIP_HEULEVEL (integer): specifies heuristic used to find integer solution←

This specifies the heuristic used to find the integer solution. Possible values are: 0: No heuristic is used. 1: A simple heuristic is used. Typically, this will find integer solutions only on problems with a certain structure. However, it tends to be fast. 2: This is an advanced heuristic that tries to find a "good" integer solution fast. In general, a value of 2 seems to not increase the total solution time and will find an integer solution fast on many problems. A higher value may find an integer solution faster, or an integer solution where none would have been found with a lower level. Try level 3 or 4 on "difficult" problems where 2 does not help. Higher values cause more time to be spent in the heuristic. The value may be set arbitrarily high. However, >20 is probably not worthwhile. MIP_HEUMINTIMLIM controls the time to be spent in searching heuristic solutions.

Default: 3

MIP_HEUMINTIMLIM (integer): minimum time in seconds to be spent in finding heuristic solutions←
This specifies the minimum time in seconds to be spent in finding heuristic solutions to the MIP model. \texttt{MIP.HEULEVEL} controls the heuristic used to find the integer solution.

Default: 0

\textbf{MIP.HEU.DROP.OBJ (integer):} flag for whether to use without OBJ heuristic

This is a flag for whether to use without OBJ heuristic.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

\textbf{MIP.HEU.MODE (integer):} heuristic used in MIP solver

This controls the MIP heuristic mode.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides when to stop the heuristic</td>
</tr>
<tr>
<td>1</td>
<td>Solver uses a pre-specified time limit to stop the heuristic.</td>
</tr>
<tr>
<td>2</td>
<td>Solver uses a pre-specified iteration limit to stop the heuristic</td>
</tr>
</tbody>
</table>

\textbf{MIP.INTTOL (real):} absolute integer feasibility tolerance

An integer variable is considered integer feasible if the absolute difference from the nearest integer is smaller than this.

Default: 1e-6

\textbf{MIP.ITRLIM (real):} iteration limit for branch and bound

This is the total LP iteration limit summed over all branches for branch-and-bound. Range for The default value is -1, which means no iteration limit is imposed. If this iteration limit is reached, branch-and-bound will stop and the best feasible integer solution found will be installed as the incumbent (best known) solution.

Range: [-1, \infty]

Default: infinity

\textbf{MIP.KBEST.USE.GOP (integer):} specifies whether to use gop solver in MIP KBest

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\textbf{MIP.KEEPINMEM (integer):} flag for keeping LP bases in memory
If this is set to 1, the integer pre-solver will try to keep LP bases in memory. This typically gives faster solution times, but uses more memory. Setting this parameter to 0 causes the pre-solver to erase bases from memory.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not keep LP bases in memory</td>
</tr>
<tr>
<td>1</td>
<td>Keep LP bases in memory</td>
</tr>
</tbody>
</table>

**MIP_LBIGM** (*real*): Big-M value used in linearizing nonlinear expressions

This refers to the Big-M value used in linearizing nonlinear expressions.

Default: 10000

**MIP_LSOLTIMLIM** (*integer*): time limit until finding a new integer solution

Range: \{-1, ..., ∞\}

Default: -1

**MIP_MAKECUT_INACTIVE_COUNT** (*integer*): threshold for times a cut could remain active after successive reoptimization

This value specifies the threshold for the times a cut could remain active after successive reoptimization during branch-and-bound. If the count is larger than the specified level the solver will inactive the cut.

Default: 10

**MIP_MAXCUTPASS_TOP** (*integer*): number passes to generate cuts on the root node

This controls the number passes to generate cuts on the root node. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: 200

**MIP_MAXCUTPASS_TREE** (*integer*): number passes to generate cuts on the child nodes

This controls the number passes to generate cuts on the child nodes. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: 2

**MIP_MAXNONIMP_CUTPASS** (*integer*): number of passes allowed in cut-generation that does not improve current relaxation

This controls the maximum number of passes allowed in cut-generation that does not improve the current relaxation.

Default: 3

**MIP_MAXNUM_MIP_SOL_STORAGE** (*integer*): maximum number of k-best solutions to store
This specifies the maximum number of k-best solutions to store. Possible values are positive integers.

Default: 1

**MIP_MINABSOBJSTEP** *(real)*: value to update cutoff value each time a mixed integer solution is found

This specifies the value to update the cutoff value each time a mixed integer solution is found.

Default: 0

**MIP_NODESELRULE** *(integer)*: specifies the node selection rule

This specifies the node selection rule for choosing between all active nodes in the branch-and-bound tree when solving integer programs. Possible selections are: 0: Solver decides (default). 1: Depth first search. 2: Choose node with worst bound. 3: Choose node with best bound. 4: Start with best bound. If no improvement in the gap between best bound and best integer solution is obtained for some time, switch to: if (number of active nodes < 10000) Best estimate node selection (5). else Worst bound node selection (2). 5: Choose the node with the best estimate, where the new objective estimate is obtained using pseudo costs. 6: Same as (4), but start with the best estimate.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Depth first search</td>
</tr>
<tr>
<td>2</td>
<td>Choose node with worst bound</td>
</tr>
<tr>
<td>3</td>
<td>Choose node with best bound</td>
</tr>
<tr>
<td>4</td>
<td>Start with best bound</td>
</tr>
<tr>
<td>5</td>
<td>Choose the node with the best estimate</td>
</tr>
<tr>
<td>6</td>
<td>Same as 4 but start with the best estimate</td>
</tr>
</tbody>
</table>

**MIP_NUM_THREADS** *(integer)*: number of parallel threads to use by the parallel MIP solver

This parameter specifies the number of parallel threads to use by the parallel MIP solver. Possible values are positive integers. The default is 1 implying that the parallel solver is disabled.

Range: \{1, \ldots, \infty\}

Default: 1

**MIP_PARA_FP** *(integer)*: flag for whether to use parallelization on the feasibility pump heuristic

This is a flag for whether to use parallelization on the feasibility pump heuristic.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>
MIP_PARA_FP_MODE (integer): flag for the mode of parallel feasibility pump

This is a flag for the mode of parallel feasibility pump.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terminate when all threads finish</td>
</tr>
<tr>
<td>1</td>
<td>Terminate as soon as the master thread finishes</td>
</tr>
</tbody>
</table>

MIP_PARA_INIT_NODE (real): number of initial nodes for MIP parallelization

This value specifies the number of initial nodes for MIP parallelization.

Range: [-1, ∞]

Default: -1

MIP_PARA_ITR_MODE (integer): flag for iteration mode in MIP parallelization

This is a flag for iteration mode in MIP parallelization.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Each thread terminates as soon as arrives iteration limit</td>
</tr>
<tr>
<td>1</td>
<td>Each thread terminates until all threads get iteration limit</td>
</tr>
</tbody>
</table>

MIP_PARA_RND_ITRLMT (real): iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations

This value specifies the iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations.

Default: 2.0

MIP_PARA_SUB (integer): flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing

This is a flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

MIP_PEROPTTOL (real): MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds

This is the MIP relative optimality tolerance that will be in effect after T seconds following the start. The value T should be specified using the MIP_AOPTTIMLIM parameter.

Default: 1e-5
**MIP_PERSPECTIVE_REFORM** *(integer)*: flag for whether to use Perspective Reformulation

This is the flag for whether to use Perspective Reformulation.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
</tr>
</tbody>
</table>

**MIP_POLISH_ALPHA_TARGET** *(real)*: proportion solutions in the pool to initiate a polishing-task at the current node

This value specifies the proportion solutions in the pool to initiate a polishing-task at the current node.

Range: [0.01, 0.99]

Default: 0.6

**MIP_POLISH_MAX_BRANCH_COUNT** *(integer)*: maximum number of branches to polish

This value specifies the maximum number of branches to polish.

Default: 2000

**MIP_POLISH_NUM_BRANCH_NEXT** *(integer)*: number of branches to polish in the next round

This value specifies the number of branches to polish in the next round.

Default: 4000

**MIP_PREHEU_DFE_VSTLIM** *(integer)*: limit for the variable visit in depth first enumeration

Limit for the variable visit in depth first enumeration.

Default: 200

**MIP_PREHEU_LEVEL** *(integer)*: heuristic level for the prerelax solver

The heuristic level for the prerelax solver.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nothing</td>
</tr>
<tr>
<td>1</td>
<td>One-change</td>
</tr>
<tr>
<td>2</td>
<td>One-change and two-change</td>
</tr>
<tr>
<td>3</td>
<td>Depth first enumeration</td>
</tr>
</tbody>
</table>

**MIP_PREHEU_TC_ITERLIM** *(integer)*: iteration limit for the two change heuristic

Iteration limit for the two change heuristic.

Default: 30000000
MIP_PREHEU_VAR_SEQ (integer): sequence of the variable considered by the prerelax heuristic

The sequence of the variable considered by the prerelax heuristic.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Backward</td>
</tr>
<tr>
<td>1</td>
<td>Forward</td>
</tr>
</tbody>
</table>

MIP_PRELEVEL (integer): controls the amount and type of MIP pre-solving at root node

This controls the amount and type of MIP pre-solving at root node.

Default: 3070

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Simple pre-solving</td>
</tr>
<tr>
<td>+4</td>
<td>Probing</td>
</tr>
<tr>
<td>+8</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+16</td>
<td>Elimination</td>
</tr>
<tr>
<td>+32</td>
<td>Dual reductions</td>
</tr>
<tr>
<td>+64</td>
<td>Use dual information</td>
</tr>
<tr>
<td>+128</td>
<td>Binary row presolving</td>
</tr>
<tr>
<td>+256</td>
<td>Row aggregation</td>
</tr>
<tr>
<td>+512</td>
<td>Coef Probe Lifting</td>
</tr>
<tr>
<td>+1024</td>
<td>Maximum pass</td>
</tr>
<tr>
<td>+2048</td>
<td>Similar row</td>
</tr>
</tbody>
</table>

MIP_PRELEVEL_TREE (integer): amount and type of MIP pre-solving at tree nodes

This controls the amount and type of MIP pre-solving at tree nodes.

Default: 1214

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Simple pre-solving</td>
</tr>
<tr>
<td>+4</td>
<td>Probing</td>
</tr>
<tr>
<td>+8</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+16</td>
<td>Elimination</td>
</tr>
<tr>
<td>+32</td>
<td>Dual reductions</td>
</tr>
<tr>
<td>+64</td>
<td>Use dual information</td>
</tr>
<tr>
<td>+128</td>
<td>Binary row presolving</td>
</tr>
<tr>
<td>+256</td>
<td>Row aggregation</td>
</tr>
<tr>
<td>+512</td>
<td>Coef Probe Lifting</td>
</tr>
<tr>
<td>+1024</td>
<td>Maximum pass</td>
</tr>
</tbody>
</table>

MIP_PRE_ELIM_FILL (integer): controls fill-in introduced by eliminations during pre-solve

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly
more than the original model.

Default: 100

**MIP_PSEUDOCOST_RULE (integer)**: specifies the rule in pseudocost computations for variable selection

This specifies the rule in pseudocost computations for variable selection.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Only use min pseudo cost</td>
</tr>
<tr>
<td>2</td>
<td>Only use max pseudo cost</td>
</tr>
<tr>
<td>3</td>
<td>Use quadratic score function and the pseudo cost weight</td>
</tr>
<tr>
<td>4</td>
<td>Same as 3 without quadratic score</td>
</tr>
</tbody>
</table>

**MIP_PSEUDOCOST_WEIGHT (real)**: weight in pseudocost computations for variable selection

This specifies the weight in pseudocost computations for variable selection.

Default: 1.5625e-05

**MIP_REDCOSTFIX_CUTOFF (real)**: cutoff value as a percentage of the reduced costs

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic.

Default: 0.9

**MIP_REDCOSTFIX_CUTOFF_TREE (real)**: cutoff value as a percentage of the reduced costs at tree nodes

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic at tree nodes.

Default: 0.9

**MIP_RELINTTOL (real)**: relative integer feasibility tolerance

An integer variable is considered integer feasible if the difference between its value and the nearest integer value divided by the value of the nearest integer is less than this.

Default: 8e-6

**MIP_RELOPTTOL (real)**: MIP relative optimality tolerance

This is the MIP relative optimality tolerance. Solutions must beat the incumbent by at least this relative amount to become the new, best solution.

Default: GAMS OptCR

**MIP_REOPT (integer)**: optimization method to use when doing reoptimization

This specifies which optimization method to use when doing reoptimization from a given basis.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use primal method</td>
</tr>
<tr>
<td>2</td>
<td>Use dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Use barrier solver</td>
</tr>
</tbody>
</table>

**MIP_SCALING_BOUND** *(integer):* maximum difference between bounds of an integer variable for enabling scaling ←

This controls the maximum difference between the upper and lower bounds of an integer variable that will enable the scaling in the simplex solver when solving a subproblem in the branch-and-bound tree.

Default: 10000

**MIP_SOLLIM** *(integer):* integer solution limit for MIP solver ←

Range: \{-1, ..., \infty\}

Default: -1

**MIP_SOLVERTYPE** *(integer):* optimization method to use when solving mixed-integer models ←

This specifies the optimization method to use when solving mixed-integer models.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use Branch and Bound only</td>
</tr>
<tr>
<td>2</td>
<td>Use Enumeration and Knapsack solver only</td>
</tr>
</tbody>
</table>

**MIP_STRONGBRANCHDONUM** *(integer):* minimum number of variables to try the strong branching on ←

This value specifies the minimum number of variables, among all the candidates, to try the strong branching on.

Default: 3

**MIP_STRONGBRANCHLEVEL** *(integer):* depth from the root in which strong branching is used ←

This specifies the depth from the root in which strong branching is used. The default value of 10 means that strong branching is used on a level of 1 to 10 measured from the root. Strong branching finds the real bound for branching on a given variable, which, in most cases, requires a solution of a linear program and may therefore also be quite expensive in computing time. However, if used on nodes close to the root node of the tree, it also gives a much better bound for that part of the tree and can therefore reduce the size of the branch-and-bound tree.

Default: 10

**MIP_SWITCHFAC_SIM_IPM_TIME** *(real):* factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method ←
This specifies the (positive) factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method. A value of -1.0 means that no time limit is imposed.

Range: \([-1, \infty]\)

Default: -1

**MIP_SYMMETRY_MODE (integer):** specifies mip symmetry handling methods

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use symmetries</td>
</tr>
<tr>
<td>1</td>
<td>Adding symmetry breaking cuts</td>
</tr>
<tr>
<td>2</td>
<td>Orbital fixing</td>
</tr>
</tbody>
</table>

**MIP_TIMLIM (integer):** time limit in seconds for integer solver

This is the time limit in seconds for branch-and-bound. The default value is -1, which means no time limit is imposed. However, the value of \(\text{SOLVER\_TIMLMT}\) will be applied to each continuous subproblem solve. If the value of this parameter is greater than 0, then the value of \(\text{SOLVER\_TIMLMT}\) will be disregarded. If this time limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: \([-1, ..., \infty]\)

Default: GAMS ResLim

**MIP_TOPOPT (integer):** optimization method to use when there is no previous basis

This specifies which optimization method to use when there is no previous basis.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use primal method</td>
</tr>
<tr>
<td>2</td>
<td>Use dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Use barrier solver</td>
</tr>
</tbody>
</table>

**MIP_TREEREORDERLEVEL (integer):** tree reordering level

This specifies the tree reordering level.

Default: 10

**MIP_TREEREORDERMODE (integer):** tree reordering mode

This specifies the tree reordering mode.

Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use tree reordering only for subproblems</td>
</tr>
<tr>
<td>2</td>
<td>Use tree reordering for subproblems and the main bnb loop only when LP status is infeasible</td>
</tr>
<tr>
<td>3</td>
<td>Not use tree reordering</td>
</tr>
<tr>
<td>4</td>
<td>Use tree reordering based on MIP_TREEREORDERLEVEL</td>
</tr>
</tbody>
</table>

**MIP_USECUTOFFOBJ** *(integer)*: flag for using branch and bound limit $\leftarrow$

This is a flag for the parameter `MIP_CUTOFFOBJ`. If you do not want to lose the value of the parameter `MIP_CUTOFFOBJ`, this provides an alternative to disabling the cutoff objective.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use current cutoff value</td>
</tr>
<tr>
<td>1</td>
<td>Use current cutoff value</td>
</tr>
</tbody>
</table>

**MIP_USE_CUTS_HEU** *(integer)*: controls if cut generation is enabled during MIP heuristics $\leftarrow$

This flag controls if cut generation is enabled during MIP heuristics. The default is -1 (i.e. the solver decides).

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Do not use cut heuristic</td>
</tr>
<tr>
<td>1</td>
<td>Use cut heuristic</td>
</tr>
</tbody>
</table>

**MIP_USE_ENUM_HEU** *(integer)*: frequency of enumeration heuristic $\leftarrow$

This specifies the frequency of enumeration heuristic.

Default: 4

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off</td>
</tr>
<tr>
<td>1</td>
<td>Only at top (root) node without cuts</td>
</tr>
<tr>
<td>2</td>
<td>Both at top (root) and tree nodes without cuts</td>
</tr>
<tr>
<td>3</td>
<td>Same as 1 with cuts</td>
</tr>
<tr>
<td>4</td>
<td>Same as 2 with cuts</td>
</tr>
</tbody>
</table>

**MIP_USE_INT_ZERO_TOL** *(integer)*: controls if all MIP calculations would be based on absolute integer feasibility tolerance $\leftarrow$

This flag controls if all MIP calculations would be based on the integrality tolerance specified by `MIP_INTTOL`.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not base MIP calculations on MIP_INTTOL</td>
</tr>
<tr>
<td>1</td>
<td>Base MIP calculations on MIP_INTTOL</td>
</tr>
</tbody>
</table>

**MULTITHREAD\_MODE** (integer): threading mode

This parameter controls the threading mode for solvers with multithreading support.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Try parallel mode but if it is not available try concurrent mode</td>
</tr>
<tr>
<td>2</td>
<td>Try parallel mode only</td>
</tr>
<tr>
<td>3</td>
<td>Try concurrent mode but if it is not available try parallel mode</td>
</tr>
<tr>
<td>4</td>
<td>Try concurrent mode only</td>
</tr>
</tbody>
</table>

**NLP\_AUTODERIV** (integer): defining type of computing derivatives

This is a flag to indicate if automatic differentiation is the method of choice for computing derivatives and select the type of differentiation.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Finite Differences approach will be used</td>
</tr>
<tr>
<td>1</td>
<td>Forward type of Automatic Differentiation will be used</td>
</tr>
<tr>
<td>2</td>
<td>Backward type of Automatic Differentiation will be used</td>
</tr>
</tbody>
</table>

**NLP\_AUTOHESS** (integer): flag for using Second Order Automatic Differentiation for solving NLP

This is a flag to indicate if Second Order Automatic Differentiation will be performed in solving a nonlinear model. The second order derivatives provide an exact/precise Hessian matrix to the SQP algorithm, which may lead to less iterations and better solutions, but may also be quite expensive in computing time for some cases.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use Second Order Automatic Differentiation</td>
</tr>
<tr>
<td>1</td>
<td>Use Second Order Automatic Differentiation</td>
</tr>
</tbody>
</table>

**NLP\_CONIC\_REFORM** (integer): determines if to explore conic reformulation

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**NLP_CUTOFFOBJ (real):** as soon as any multi-start thread achieves this value all threads stop

If the current best objective of the NLP being solved in a multistart run is better than this value, the solver will terminate early without exhausting the maximum number of multistarts. This is a way of saving computer time if the current best solution is sufficiently attractive.

Range: \([-1e30, \infty]\)

Default: \(-1e30\)

**NLP_DERIV_DIFFTYPE (integer):** flag indicating the technique used in computing derivatives with finite differences

This is a flag indicating the technique used in computing derivatives with Finite Differences.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use forward differencing method</td>
</tr>
<tr>
<td>2</td>
<td>Use backward differencing method</td>
</tr>
<tr>
<td>3</td>
<td>Use center differencing method</td>
</tr>
</tbody>
</table>

**NLP_FEASCHK (integer):** how to report results when solution satisfies tolerance of scaled but not original model

This input parameter specifies how the NLP solver reports the results when an optimal or local-optimal solution satisfies the feasibility tolerance (NLP_FEASTOL) of the scaled model but not the original (descaled) one.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Perform no action accept the final solution</td>
</tr>
<tr>
<td>1</td>
<td>Declare the model status as FEASIBLE if maximum violation in the unscaled model is not higher than 10 times NLP_FEASTOL</td>
</tr>
<tr>
<td>2</td>
<td>Declare the model status as UNKNOWN if maximum violation in the unscaled model is higher than NLP_FEASTOL</td>
</tr>
</tbody>
</table>

**NLP_FEASTOL (real):** feasibility tolerance for nonlinear constraints

This is the feasibility tolerance for nonlinear constraints. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: \(1e-6\)

**NLP_INF (real):** numeric infinity for nonlinear models

Specifies the numeric infinity for nonlinear models. Possible values are positive real numbers. Smaller values could cause numerical problems.

nlp_ipm2grg This is a flag to switch from IPM solver to the standard NLP (GRG) solver when IPM fails due to numerical errors.

Default: \(1e30\)
**NLP_IPM2GRG** (integer): switch from IPM solver to GRG solver when IPM fails due to numerical errors

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not switch</td>
</tr>
<tr>
<td>1</td>
<td>Switch</td>
</tr>
</tbody>
</table>

**NLP_ITERS_PER_LOGLINE** (integer): number of nonlinear iterations to elapse before next progress message

Number of nonlinear iterations to elapse before next progress message.

Range: \( \{1, \ldots, \infty\} \)

Default: 10

**NLP_ITRLMT** (integer): nonlinear iteration limit

This controls the iteration limit on the number of nonlinear iterations performed.

Range: \( \{-1, \ldots, \infty\} \)

Default: GAMS IterLim

**NLP_LINEARZ** (integer): extent to which the solver will attempt to linearize nonlinear models

This determines the extent to which the solver will attempt to linearize nonlinear models.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>No linearization occurs</td>
</tr>
<tr>
<td>2</td>
<td>Linearize ABS MAX and MIN functions</td>
</tr>
<tr>
<td>3</td>
<td>Same as option 2 plus IF AND OR NOT and all logical operators are linearized</td>
</tr>
</tbody>
</table>

**NLP_LINEARZ_WBCONSISTENT** (integer): determines if linearization process is consistent with WB/excel calculation

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NLP_MAXLOCALSEARCH** (integer): maximum number of local searches

This controls the maximum number of local searches (multistarts) when solving a NLP using the multistart solver.

Default: 5
**NLP_MAXLOCALSEARCH_TREE** (integer): maximum number of multistarts

Maximum number of multistarts (at tree nodes)

Default: 1

**NLP_MAX_RETRY** (integer): maximum number refinement retries to purify the final NLP solution

Maximum number refinement retries to purify the final NLP solution.

Range: \{-1, ..., \infty\}

Default: 5

**NLP_MSW_EUCDIST_THRES** (real): euclidean distance threshold in multistart search

Euclidean distance threshold in multistart search

Default: 0.001

**NLP_MSW_FILTMODE** (integer): filtering mode to exclude certain domains during sampling in multistart search

Filtering mode to exclude certain domains during sampling in multistart search.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>+1</td>
<td>Filter-out the points around known KKT or feasible points previously visited</td>
</tr>
<tr>
<td>+2</td>
<td>Filter-out the points whose p are in the vicinity of p(x)</td>
</tr>
<tr>
<td>+4</td>
<td>Filter-out the points in the vicinity of x where x are initial points of all previous local optimizations</td>
</tr>
<tr>
<td>+8</td>
<td>Filter-out the points whose p(.) values are below a dynamic threshold tolerance</td>
</tr>
</tbody>
</table>

**NLP_MSW_MAXNOIMP** (integer): maximum number of consecutive populations to generate without any improvements

Maximum number of consecutive populations to generate without any improvements.

Range: \{-1, ..., \infty\}

Default: -1

**NLP_MSW_MAXPOP** (integer): maximum number of populations to generate in multistart search

Maximum number of populations to generate in multistart search.

Range: \{-1, ..., \infty\}

Default: -1

**NLP_MSW_MAXREF** (integer): maximum number of reference points to generate trial points in multistart search

Maximum number of reference points to generate trial points in multistart search.
Maximum number of reference points in the solution space to generate trial points in multistart search.

Range: \{-1, ..., \infty\}

Default: -1

**NLP_MSW_NORM** *(integer)*: norm to measure the distance between two points in multistart search

Norm to measure the distance between two points in multistart search.

Range: \{-1, ..., \infty\}

Default: 2

**NLP_MSW_NUM_THREADS** *(integer)*: number of parallel threads to be used when solving an NLP model with the multistart solver

This value specifies the number of parallel threads to be used when solving an NLP model with the multistart solver.

Default: 1

**NLP_MSW_OVERLAP_RATIO** *(real)*: rate of replacement in successive populations

This value specifies the rate of replacement in successive populations. Higher values favours survival of points in the parent population.

Range: \([0.0, 1.0]\)

Default: 0.1

**NLP_MSW_POXDIST THRES** *(real)*: penalty function neighborhood threshold in multistart search

Penalty function neighborhood threshold in multistart search

Default: 0.01

**NLP_MSW_PREPMODE** *(integer)*: preprocessing strategies in multistart solver

This value specifies the preprocessing strategies in multistart solver.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>+1</td>
<td>Truncate free variables</td>
</tr>
<tr>
<td>+2</td>
<td>Scale reference points to origin</td>
</tr>
<tr>
<td>+4</td>
<td>Enable expansive scaling of radius[k] by hit[k]</td>
</tr>
<tr>
<td>+8</td>
<td>Skewed sampling allowing values in the vicinity of origin.</td>
</tr>
<tr>
<td>+16</td>
<td>Get best bounds by presolver</td>
</tr>
<tr>
<td>+32</td>
<td>Get best bounds using GOP</td>
</tr>
<tr>
<td>+64</td>
<td>Enable sampling of free variables (not recommended)</td>
</tr>
<tr>
<td>+128</td>
<td>Collect sufficiently many trial points prior to local solves</td>
</tr>
<tr>
<td>+256</td>
<td>Enable power solver, trying several different local strategies</td>
</tr>
</tbody>
</table>
**NLP_MSW_RG_SEED** (*integer*): random number generator seed for the multistart solver

This value specified the random number generator seed for the multistart solver.

Default: 1019

**NLP_MSW_SOLIDX** (*integer*): index of the multistart solution to be loaded

Index of the multistart solution to be loaded main solution structures.

Default: 0

**NLP_MSW_XKKTRAD_FACTOR** (*real*): KKT solution neighborhood factor in multistart search

KKT solution neighborhood factor in multistart search

Default: 0.85

**NLP_MSW_XNULRAD_FACTOR** (*real*): initial solution neighborhood factor in multistart search

Initial solution neighborhood factor in multistart search

Default: 0.5

**NLP_PRELEVEL** (*integer*): controls the amount and type of NLP pre-solving

This controls the amount and type of NLP pre-solving.

Default: 126

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Simple pre-solving</td>
</tr>
<tr>
<td>+4</td>
<td>Probing</td>
</tr>
<tr>
<td>+8</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+16</td>
<td>Elimination</td>
</tr>
<tr>
<td>+32</td>
<td>Dual reductions</td>
</tr>
<tr>
<td>+64</td>
<td>Use dual information</td>
</tr>
<tr>
<td>+512</td>
<td>Maximum pass</td>
</tr>
</tbody>
</table>

**NLP_PSTEP_FINITEDIFF** (*real*): value of the step length in computing the derivatives using finite differences

This controls the value of the step length in computing the derivatives using finite differences.

Default: 5e-7

**NLP_QUADCHK** (*integer*): flag for checking if NLP is quadratic

This is a flag indicating if the nonlinear model should be examined to check if it is a quadratic model.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not check if NLP is quadratic</td>
</tr>
<tr>
<td>1</td>
<td>Check if NLP is quadratic</td>
</tr>
</tbody>
</table>
**NLP_REDTOL** (*real*): tolerance for the gradients of nonlinear functions

This is the tolerance for the gradients of nonlinear functions. The (projected) gradient of a function is considered to be the zero-vector if its norm is below this tolerance.

Default: $1 \text{e}^{-7}$

**NLP_SOLVER** (*integer*): type of nonlinear solver

This value determines the type of nonlinear solver.

Default: 7

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Solver decides</td>
</tr>
<tr>
<td>5</td>
<td>Uses Levenberg-Marquardt method to solve nonlinear least-squares problem</td>
</tr>
<tr>
<td>6</td>
<td>Uses Barrier solver for convex QCP models</td>
</tr>
<tr>
<td>7</td>
<td>Uses CONOPT's reduced gradient solver</td>
</tr>
<tr>
<td>8</td>
<td>Uses SLP solver</td>
</tr>
<tr>
<td>9</td>
<td>Uses CONOPT with multistart feature enabled</td>
</tr>
</tbody>
</table>

**NLP_SOLVE_AS_LP** (*integer*): flag indicating if the nonlinear model will be solved as an LP

This is a flag indicating if the nonlinear model will be solved as an LP. 1 means that an LP using first order approximations of the nonlinear terms in the model will be used when optimizing the model with the LSoptimize() function.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NLP will not be solved as LP</td>
</tr>
<tr>
<td>1</td>
<td>NLP will be solved as LP</td>
</tr>
</tbody>
</table>

**NLPSTALLITRLMT** (*integer*): iteration limit before a sequence of non-improving NLP iterations is declared as stalling

This specifies the iteration limit before a sequence of non-improving NLP iterations is declared as stalling, thus causing the solver to terminate.

Default: 100

**NLP_STARTPOINT** (*integer*): flag for using initial starting solution for NLP

This is a flag indicating if the nonlinear solver should accept initial starting solutions.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use initial starting solution for NLP</td>
</tr>
<tr>
<td>1</td>
<td>Use initial starting solution for NLP</td>
</tr>
</tbody>
</table>

**NLP_SUBSOLVER** (*integer*): type of nonlinear subsolver
This controls the type of linear solver to be used for solving linear subproblems when solving nonlinear models.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primal simplex method</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex method</td>
</tr>
<tr>
<td>3</td>
<td>Barrier solver with or without crossover</td>
</tr>
</tbody>
</table>

**NLP_USECUTOFFOBJ** *(integer)*: flag to use parameter NLP_CUTOFFOBJ

This is a flag for the parameter NLP_CUTOFFOBJ. The value of 0 means NLP_CUTOFFOBJ will be ignored, else it will be used as specified.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NLP_USE_CRASH** *(integer)*: flag for using simple crash routines for initial solution

This is a flag indicating if an initial solution will be computed using simple crash routines.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use simple crash routines</td>
</tr>
<tr>
<td>1</td>
<td>Use simple crash routines</td>
</tr>
</tbody>
</table>

**NLP_USE_LINDO_CRASH** *(integer)*: flag for using advanced crash routines for initial solution

This is a flag indicating if an initial solution will be computed using advanced crash routines.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use advanced crash routines</td>
</tr>
<tr>
<td>1</td>
<td>Use advanced crash routines</td>
</tr>
</tbody>
</table>

**NLP_USE_SDP** *(integer)*: flag to use SDP solver for POSD constraint

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>
NLP_USE_SELCON(eval) (integer): flag for using selective constraint evaluations for solving NLP

This is a flag indicating if selective constraint evaluations will be performed in solving a nonlinear model.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use selective constraint evaluations</td>
</tr>
<tr>
<td>1</td>
<td>Use selective constraint evaluations</td>
</tr>
</tbody>
</table>

NLP_USE_SLP (integer): flag for using sequential linear programming step directions for updating solution

This is a flag indicating if sequential linear programming step directions should be used in updating the solution.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Do not use sequential linear programming step directions</td>
</tr>
<tr>
<td>1</td>
<td>Use sequential linear programming step directions</td>
</tr>
</tbody>
</table>

NLP_USE_STEEPEDGE (integer): flag for using steepest edge directions for updating solution

This is a flag indicating if steepest edge directions should be used in updating the solution.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use steepest edge directions</td>
</tr>
<tr>
<td>1</td>
<td>Use steepest edge directions</td>
</tr>
</tbody>
</table>

NUM_THREADS (integer): number of parallel threads to be used

Synonym: gthreads

Number of threads to use in the solver routine to be called. It is a solver-independent parameter which internally sets solver-specific threading parameters automatically.

Range: \{1, ..., \infty\}

Default: GAMS Threads

PROB_TO_SOLVE (integer): controls whether the explicit primal or dual form of the given LP problem will be solved

This flag controls whether the explicit primal or dual form of the given LP problem will be solved.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Explicit primal form</td>
</tr>
<tr>
<td>2</td>
<td>Explicit dual form</td>
</tr>
</tbody>
</table>

**PROFILER_LEVEL (integer):** specifies the profiler level to break down the total cpu time into.

Specifies the profiler level to break down the total cpu time into.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Profiler is off</td>
</tr>
<tr>
<td>+1</td>
<td>Enable for simplex solver</td>
</tr>
<tr>
<td>+2</td>
<td>Enable for integer solver</td>
</tr>
<tr>
<td>+4</td>
<td>Enable for multistart solver</td>
</tr>
<tr>
<td>+8</td>
<td>Enable for global solver</td>
</tr>
</tbody>
</table>

**READPARAMS (string):** read Lindo parameter file

**REPORTEVSOl (no value):** solve and report the expected value solution

Default: 0

**SAMP_CDSINC (real):** correlation matrix diagonal shift increment

Correlation matrix diagonal shift increment.

Default: 1e-6

**SAMP_NCM_CUTOBJ (real):** objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem

Objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem.

Default: 1e-30

**SAMP_NCM_DSTORAGE (integer):** flag to enable or disable sparse mode in NCM computations

Flag to enable/disble sparse mode in NCM computations.

Range: \{-1, ..., ∞\}

Default: -1

**SAMP_NCM_ITERLIM (integer):** iteration limit for NCM method

Iteration limit for NCM method.

Default: 100

**SAMP_NCM_METHOD (integer):** bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem

---

{5.24 LINDO and LINDOGlobal 2021}
Bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem.

Default: 5

**SAMP_NCM_OPTTOL (real):** optimality tolerance for NCM method

Optimality tolerance for NCM method.

Default: 1e-7

**SAMP_SCALE (integer):** flag to enable scaling of raw sample data

Flag to enable scaling of raw sample data.

Default: 0

**SOLVER_CONCURRENT_OPTMODE (integer):** controls if simplex and interior-point optimizers will run concurrently

Controls if simplex and interior-point optimizers will run concurrently, 0 means no concurrent runs will be performed, 1 means both optimizers will run concurrently if at least two threads exist in system, 2 means both optimizers will run concurrently.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no concurrent runs</td>
</tr>
<tr>
<td>1</td>
<td>run concurrently if at least 2 threads exist</td>
</tr>
<tr>
<td>2</td>
<td>run concurrently</td>
</tr>
</tbody>
</table>

**SOLVER_CUTOFFVAL (real):** solver will exit if optimal solution is worse than this

If the optimal objective value of the LP being solved is shown to be worse than this (e.g., if the dual simplex method is being used), then the solver will exit without finding a feasible solution. This is a way of saving computer time if there is no sufficiently attractive solution. **SOLVER_USECUTOFFVAL** needs to be set to 1 to activate this value.

Default: 0

**SOLVER_FEASTOL (real):** feasibility tolerance

This is the feasibility tolerance. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: 1e-7

**SOLVER_IPMSOL (integer):** basis crossover flag for barrier solver

This flag controls whether a basis crossover will be performed when solving LPs with the barrier solver. A value of 0 indicates that a crossover to a basic solution will be performed. If the value is 1, then the barrier solution will be left intact. For example, if alternate optima exist, the barrier method will return a solution that is, loosely speaking, the average of all alternate optima.

Default: 0
### Value | Meaning
--- | ---
0 | Perform crossover to basis solution
1 | Leave barrier solution intact

**SOLVER_IUSOL** (*integer*): flag for computing basic solution for infeasible model

This is a flag that, when set to 1, will force the solver to compute a basic solution to an infeasible model that minimizes the sum of infeasibilities and a basic feasible solution to an unbounded problem from which an extreme direction originates. When set to the default of 0, the solver will return with an appropriate status flag as soon as infeasibility or unboundedness is detected. If infeasibility or unboundedness is declared with presolver’s determination, no solution will be computed.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Return appropriate status if infeasibility is encountered</td>
</tr>
<tr>
<td>1</td>
<td>Force the solver to compute a basic solution to an infeasible model</td>
</tr>
</tbody>
</table>

**SOLVER_METHOD** (*integer*): specifies the method to use when generic solver is invoked

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FREE</td>
</tr>
<tr>
<td>1</td>
<td>PSIMPLEX</td>
</tr>
<tr>
<td>2</td>
<td>DSIMPLEX</td>
</tr>
<tr>
<td>3</td>
<td>BARRIER</td>
</tr>
<tr>
<td>4</td>
<td>NLP</td>
</tr>
<tr>
<td>5</td>
<td>MIP</td>
</tr>
<tr>
<td>6</td>
<td>MULTIS</td>
</tr>
<tr>
<td>7</td>
<td>GOP</td>
</tr>
<tr>
<td>8</td>
<td>IIS</td>
</tr>
<tr>
<td>9</td>
<td>IUS</td>
</tr>
<tr>
<td>10</td>
<td>SBD</td>
</tr>
<tr>
<td>11</td>
<td>SPRINT</td>
</tr>
<tr>
<td>12</td>
<td>GA</td>
</tr>
</tbody>
</table>

**SOLVER_MODE** (*integer*): controls some of the advanced strategies when solving LPs

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>add distinct basic solutions to the pool of alternate optimal solutions</td>
</tr>
<tr>
<td>+2</td>
<td>add edge/nonbasic solutions to the pool of alternate optimal solutions</td>
</tr>
<tr>
<td>+4</td>
<td>favor basic solutions with integer values when choosing solutions to add to the pool of alternate optimal solutions</td>
</tr>
</tbody>
</table>

**SOLVER_OPTTOL** (*real*): dual feasibility tolerance
This is the optimality tolerance. It is also referred to as the dual feasibility tolerance. A dual slack (reduced cost) is considered violated if it violates its lower bound by the optimality tolerance.

Default: 1e-7

**SOLVER_PRE_ELIM_FILL** *(integer)*: fill-in introduced by the eliminations during pre-solve

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.

Default: 1000

**SOLVER_RESTART** *(integer)*: starting basis flag

This is the starting basis flag. 1 means LINDO API will perform warm starts using any basis currently in memory. 0 means LINDO API will perform cold starts discarding any basis in memory and starting from scratch.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Perform cold start</td>
</tr>
<tr>
<td>1</td>
<td>Perform warm start</td>
</tr>
</tbody>
</table>

**SOLVER_TIMLMT** *(integer)*: time limit in seconds for continous solver

This is a time limit in seconds for the LP solver. The default value of -1 imposes no time limit.

Range: \{-1, ..., ∞\}

Default: GAMS ResLim

**SOLVER_USECUTOFFVAL** *(integer)*: flag for using cutoff value

This is a flag for the parameter **SOLVER_CUTOFFVAL**

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use cutoff value</td>
</tr>
<tr>
<td>1</td>
<td>Use cutoff value</td>
</tr>
</tbody>
</table>

**SPLEX_DPRICING** *(integer)*: pricing option for dual simplex method

This is the pricing option to be used by the dual simplex method.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides the dual pricing method</td>
</tr>
<tr>
<td>0</td>
<td>Partial pricing</td>
</tr>
<tr>
<td>1</td>
<td>Steepest edge</td>
</tr>
</tbody>
</table>
**SPLEX_DUAL_PHASE** *(integer)*: controls the dual simplex strategy

This controls the dual simplex strategy, single-phase versus two-phase.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Single-phase</td>
</tr>
<tr>
<td>2</td>
<td>Two-phase</td>
</tr>
</tbody>
</table>

**SPLEX_PPRICING** *(integer)*: pricing option for primal simplex method

This is the pricing option to be used by the primal simplex method.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides the primal pricing method</td>
</tr>
<tr>
<td>0</td>
<td>Partial pricing</td>
</tr>
<tr>
<td>1</td>
<td>Devex</td>
</tr>
</tbody>
</table>

**SPLEX_REFACFRQ** *(integer)*: number of simplex iterations between two consecutive basis refactorizations

This is a positive integer scalar referring to the simplex iterations between two consecutive basis refactorizations. For numerically unstable models, setting this parameter to smaller values may help.

Default: 100

**STOC_ABSOPTTOL** *(real)*: absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver

Absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.

Default: GAMS OptCA

**STOC_ADD_MPI** *(integer)*: flag to use add-instructions mode when building deteq

Flag to use add-instructions mode when building deteq.

Default: 0

**STOC_ALD_DUAL_FEASTOL** *(real)*: dual feasibility tolerance for ALD

Dual feasibility tolerance for ALD.

Default: 1e-4

**STOC_ALD_DUAL_STEPLEN** *(real)*: dual step length for ALD
Dual step length for ALD.
Default: 0.9

**STOC_ALD_INNER_ITER_LIM** (integer): inner loop iteration limit for ALD

Inner loop iteration limit for ALD.
Default: 1000

**STOC_ALD_OUTER_ITER_LIM** (integer): outer loop iteration limit for ALD

Outer loop iteration limit for ALD.
Default: 200

**STOC_ALD_PRIMAL_FEASTOL** (real): primal feasibility tolerance for ALD

Primal feasibility tolerance for ALD.
Default: 1e-4

**STOC_ALD_PRIMAL_STEPLEN** (real): primal step length for ALD

Primal step length for ALD.
Default: 0.5

**STOC_AUTOAGG** (integer): flag to enable or disable autoaggregation

Flag to enable or disable autoaggregation.
Default: 1

**STOC_BENCHMARK_SCEN** (integer): benchmark scenario to compare EVPI and EVMU against

Benchmark scenario to compare EVPI and EVMU against.
Range: \([-2, ..., \infty]\)
Default: -2

**STOC_BIGM** (real): big-M value for linearization and penalty functions

Big-M value for linearization and penalty functions.
Default: 1e7

**STOC_BUCKET_SIZE** (integer): bucket size in Benders decomposition

Bucket size in Benders decomposition. Possible values are positive integers or (-1) for solver decides.
Range: \([-1, ..., \infty]\)
Default: -1

**STOC_CALC_EVPI** (integer): flag to enable or disable calculation of EVPI

Flag to enable/disable calculation of lower bounds on EVPI.
Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disable</td>
</tr>
<tr>
<td>1</td>
<td>enable</td>
</tr>
</tbody>
</table>

STOC\_CORRELATION\_TYPE (integer): correlation type associated with correlation matrix

Correlation type associated with the correlation matrix.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Target correlation</td>
</tr>
<tr>
<td>0</td>
<td>Pearson correlation</td>
</tr>
<tr>
<td>1</td>
<td>Kendall correlation</td>
</tr>
<tr>
<td>2</td>
<td>Spearman correlation</td>
</tr>
</tbody>
</table>

STOC\_DEQOPT (integer): method to solve the DETEQ problem

This specifies the method to use when solving the deterministic equivalent.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>10</td>
<td>Use simple Benders Decomposition</td>
</tr>
</tbody>
</table>

STOC\_DETEQ\_TYPE (integer): type of deterministic equivalent

Type of deterministic equivalent to be used by the solver. Implicit determinisitic equivalent is valid for linear and integer models only.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Implicit determinisitic equivalent</td>
</tr>
<tr>
<td>1</td>
<td>Explicit determinisitic equivalent</td>
</tr>
</tbody>
</table>

STOC\_DS\_SUBFORM (integer): subproblem formulation to use in DirectSearch

This parameter specifies the type of subproblem formulation to be used in heuristic search.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solver decides</td>
</tr>
<tr>
<td>0</td>
<td>Perform heuristic search in the original solution space</td>
</tr>
<tr>
<td>1</td>
<td>Perform heuristic search in the space of discrete variables coupled with optimizations in the linear space</td>
</tr>
</tbody>
</table>
**STOC_ELIM_FXVAR** *(integer)*: flag to enable elimination of fixed variables from deteq MPI

Flag to enable elimination of fixed variables from deteq MPI.

Default: 1

**STOC_INFBND** *(real)*: value to truncate infinite bounds at non-leaf nodes

Value to truncate infinite bounds at nonleaf nodes.

Default: 1e9

**STOC_ITER_LIM** *(integer)*: iteration limit for stochastic solver

Iteration limit for stochastic solver. Possible values are positive integers or (-1) no limit.

Range: \{-1, ..., \infty\}

Default: infinity

**STOC_MAP_MPI2LP** *(integer)*: flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements

Flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements.

Default: 0

**STOC_MAX_NUMSCENS** *(integer)*: maximum number of scenarios before forcing automatic sampling

Maximum number of scenarios before forcing automatic sampling. Possible values are positive integers.

Default: 40000

**STOC_METHOD** *(integer)*: stochastic optimization method to solve the model

Stochastic optimization method to solve the model.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Solve with the method chosen by the solver</td>
</tr>
<tr>
<td>0</td>
<td>Solve the deterministic equivalent (DETEQ)</td>
</tr>
<tr>
<td>1</td>
<td>Solve with the Nested Benders Decomposition (NBD) method</td>
</tr>
</tbody>
</table>

**STOC_NAMEDATA_LEVEL** *(integer)*: name data level

Name data level.

Default: 1

**STOC_NODELP_PRELEVEL** *(integer)*: presolve level solving node-models

Presolve level solving node-models.

Default: 0
### STOC\_NSAMPLE\_PER\_STAGE (string): list of sample sizes per stage (starting at stage 2)

Comma separated list of sample sizes per stage. The sample size of stage 1 is assumed to be 1 so that this list starts with stage 2.

### STOC\_NSAMPLE\_SPAR (integer): common sample size per stochastic parameter

Common sample size per stochastic parameter. Possible values are positive integers.

Range: \{-1, ..., \infty\}

Default: -1

### STOC\_NSAMPLE\_STAGE (integer): common sample size per stage

Common sample size per stage.

Range: \{-1, ..., \infty\}

Default: -1

### STOC\_NUM\_THREADS (integer): number of parallel threads

This value specifies the number of parallel threads to be used when solving a stochastic programming model.

Default: 1

### STOC\_RELOPTTOL (real): relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver

Relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in \( (0,1) \) interval.

Default: GAMS OptCR

### STOC\_REL\_DSTEPTOL (real): dual-step tolerance

This value specifies the dual-step tolerance in decomposition based algorithms.

Default: 1e-7

### STOC\_REL\_PSTEPTOL (real): primal-step tolerance

This value specifies the primal-step tolerance in decomposition based algorithms.

Default: 1e-8

### STOC\_REOPT (integer): reoptimization method to solve the node-models

Reoptimization method to solve the node-models.

Default: 0

---

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>Simple pre-solving</td>
</tr>
<tr>
<td>+4</td>
<td>Probing</td>
</tr>
<tr>
<td>+8</td>
<td>Coefficient reduction</td>
</tr>
<tr>
<td>+16</td>
<td>Elimination</td>
</tr>
<tr>
<td>+32</td>
<td>Dual reductions</td>
</tr>
<tr>
<td>+64</td>
<td>Use dual information</td>
</tr>
<tr>
<td>+512</td>
<td>Maximum pass</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use primal method</td>
</tr>
<tr>
<td>2</td>
<td>Use dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Use barrier solver</td>
</tr>
<tr>
<td>4</td>
<td>Use NLP solver</td>
</tr>
</tbody>
</table>

**STOC_RG_SEED** *(integer)*: seed to initialize the random number generator

Seed to initialize the random number generator. Possible values are positive integers.

Default: 1031

**STOC_SAMP_CONT_ONLY** *(integer)*: flag to restrict sampling to continuous stochastic parameters only or not

Flag to restrict sampling to continuous stochastic parameters only or not.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disable</td>
</tr>
<tr>
<td>1</td>
<td>enable</td>
</tr>
</tbody>
</table>

**STOC_SBD_MAXCUTS** *(integer)*: max cuts to generate for master problem

Max cuts to generate for master problem.

Range: \{-1, \ldots, \infty\}

Default: -1

**STOC_SBD_NUMCANDID** *(integer)*: maximum number of candidate solutions to generate at SBD root

Maximum number of candidate solutions to generate at SBD root.

Range: \{-1, \ldots, \infty\}

Default: -1

**STOC_SBD_OBJCUTFLAG** *(integer)*: flag to enable objective cut in SBD master problem

Flag to enable objective cut in SBD master problem.

Default: 1

**STOC_SBD_OBJCUTVAL** *(real)*: RHS value of objective cut in SBD master problem

RHS value of objective cut in SBD master problem.

Default: 1e-30

**STOC_SHARE_BEGSTAGE** *(integer)*: stage beyond which node-models are shared


Stage beyond which node-models share the same model structure. Possible values are positive integers less than or equal to number of stages in the model or \((-1)\) for solver decides.

Range: \([-1, \, \infty]\)

Default: \(-1\)

\textbf{STOC\_TIME\_LIM} \textit{(real)}: time limit for stochastic solver

Time limit for stochastic solver. Possible values are nonnegative real numbers or -1 for solver decides.

Range: \([-1, \, \infty]\)

Default: GAMS ResLim

\textbf{STOC\_TOPOPT} \textit{(integer)}: optimization method to solve the root problem

Optimization method to solve the root problem.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Solver decides</td>
</tr>
<tr>
<td>1</td>
<td>Use primal method</td>
</tr>
<tr>
<td>2</td>
<td>Use dual simplex</td>
</tr>
<tr>
<td>3</td>
<td>Use barrier solver</td>
</tr>
<tr>
<td>4</td>
<td>Use NLP solver</td>
</tr>
<tr>
<td>6</td>
<td>Use multi-start solver</td>
</tr>
<tr>
<td>7</td>
<td>Use global solver</td>
</tr>
</tbody>
</table>

\textbf{STOC\_VARCONTROL\_METHOD} \textit{(integer)}: sampling method for variance reduction

Sampling method for variance reduction.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Montecarlo sampling</td>
</tr>
<tr>
<td>1</td>
<td>Latinsquare sampling</td>
</tr>
<tr>
<td>2</td>
<td>Antithetic sampling</td>
</tr>
</tbody>
</table>

\textbf{STOC\_WSBAS} \textit{(integer)}: warm start basis for wait-see model

Warm start basis for wait-see model.

Range: \([-1, \, \infty]\)

Default: -1

\textbf{SVR\_LS\_ANTITHETIC} \textit{(string)}: Sample variance reduction map to Lindo Antithetic algorithm
SVR_LS_LATINSQUARE (string): Sample variance reduction map to Lindo Latin Square algorithm

SVR_LS_MONTECARLO (string): Sample variance reduction map to Lindo Montecarlo algorithm

TUNER_PRINT_LEVEL (integer): specifies the amount of print to do during parameter tuning

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not print anything default</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Print more information</td>
</tr>
</tbody>
</table>

USEGOP (integer): use global optimization

This value determines whether the global optimization will be used.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use global optimization</td>
</tr>
<tr>
<td>1</td>
<td>Use global optimization</td>
</tr>
</tbody>
</table>

WRITEDEMPI (string): write deterministic equivalent in MPI format

WRITEDEMPS (string): write deterministic equivalent in MPS format

WRITEMPI (string): write (S)MPI file of processed model

If this option is set, Lindo write an MPI file of processed model. If set, the value of this option defines the name of the MPI file.

WRITEMPS (string): write (S)MPS file of processed model

5.24.8 Stochastic Programming (SP) in GAMS/Lindo

GAMS/Lindo can also solve stochastic programming models. The syntax to set up an SP problem in GAMS is explained in the chapter Stochastic Programming (SP) with EMP. The options to control LINDO's stochastic solver are described in the subsection SP Options.
5.25 MILES

Thomas F. Rutherford, University of Colorado

5.25.1 Abstract

MILES is a solver for nonlinear complementarity problems and nonlinear systems of equations. The solver can be accessed through GAMS to solve MPSGE or MCP models. This paper documents the solution algorithm, user options, and solver output. The purpose of the paper is to provide users of MILES an overview of how the solver works so that they can use it effectively.

5.25.2 Introduction

MILES is a Fortran-based solver for nonlinear complementarity problems and nonlinear systems of equations. The solution procedure is a generalized Newton method with a backtracking line search. This code is based on an algorithm investigated by Mathiesen (1985) who proposed a modeling format and sequential method for solving economic equilibrium models. The method is closely related to algorithms proposed by Robinson (1975), Hogan (1977), Eaves (1978) and Josephy (1979). In this implementation, subproblems are solved as linear complementarity problems (LCPs), using an extension of Lemke’s almost-complementary pivoting scheme in which upper and lower bounds are represented implicitly. The linear solver employs the basis factorization package LUSOL, developed by Gill et al. (1991).

The class of problems for which MILES may be applied are referred to as ”generalized” or ”mixed” complementarity problems, which is defined as follows:

Given: \( F : \mathbb{R}^n \rightarrow \mathbb{R}^n \), \( \ell, u \in \mathbb{R}^n \)

Find: \( z, w, v \in \mathbb{R}^n \)

such that \( F(z) = w - v \)
\( \ell \leq z \leq u, \ w \geq 0, \ v \geq 0 \)
\( w^T(z - \ell) = 0 \), \( v^T(u - z) = 0 \).

When \( \ell = -\infty \) and \( u = \infty \) MCP reduces to a nonlinear system of equations. When \( \ell = 0 \) and \( u = +\infty \), the MCP is a nonlinear complementarity problem. Finite dimensional variational inequalities are also MCP. MCP includes inequality-constrained linear, quadratic and nonlinear programs as special cases, although for these problems standard optimization methods may be preferred. MCP models which are not optimization problems encompass a large class of interesting mathematical programs. Specific examples of MCP formulations are not provided here. See Rutherford (1992a) for MCP formulations arising in economics. Other examples are provided by Harker and Pang (1990) and Dirkse (1993).

There are two types of GAMS models which can be presented to MILES:

1. MILES may be used to solve computable general equilibrium models generated by MPSGE as a GAMS subsystem. In the MPSGE language, a model-builder specifies classes of nonlinear functions using a specialized tabular input format embedded within a GAMS program. Using benchmark quantities and prices, MPSGE automatically calibrates function coefficients and generates nonlinear equations and Jacobian matrices. Large, complicated systems of nonlinear equations may be implemented and analyzed very easily using this interface to MILES. An introduction to general equilibrium modeling with GAMS/MPSGE is provided by Rutherford (1992a).
2. MILES may be accessed as a GAMS solver for mixed complementarity problems (MCP). If more than one MCP solver is available \(^3\), the statement `OPTION MCP = MILES;` tells GAMS to use MILES as the MCP solution system. When problems are presented to MILES using the MCP format, the user specifies nonlinear functions using GAMS matrix algebra and the GAMS compiler automatically generates the Jacobian functions. An introduction to the GAMS/MCP modeling format is provided by Rutherford (1992b).

The purpose of this document is to provide users of MILES with an overview of how the solver works so that they can use the program more effectively. Section The Newton Algorithm introduces the Newton algorithm. Section Lemke’s Method with Implicit Bounds describes the implementation of Lemke’s algorithm which is used to solve linear subproblems. Section The Options File defines switches and tolerances which may be specified using the options file. Section Log File Output interprets the run-time log file which is normally directed to the screen. Section Status File Output interprets the status file and the detailed iteration reports which may be generated. Section Termination Messages lists and suggests remedies for abnormal termination conditions.

5.25.3 The Newton Algorithm

The iterative procedure applied within MILES to solve nonlinear complementarity problems is closely related to the classical Newton algorithm for nonlinear equations. This first part of this section reviews the classical procedure. A thorough introduction to these ideas is provided by Dennis and Schnabel (1983). For a practical perspective, see Press et al. (1986).

Newton algorithms for nonlinear equations begin with a local (Taylor series) approximation of the system of nonlinear equations. For a point \( z \) in the neighborhood of \( \bar{z} \), the system of nonlinear functions is linearized:

\[
LF(z) = F(\bar{z}) + \nabla F(\bar{z})(z - \bar{z}).
\]

Solving the linear system \( LF(z) = 0 \) provides the Newton direction from \( \bar{z} \) which is given by \( d = -\nabla F^{-1} F(\bar{z}) \).

Newton iteration \( k \) begins at point \( z^k \). First, the linear model formed at \( z^k \) is solved to determine the associated "Newton direction", \( d^k \). Second, a line search in direction \( d^k \) determines the scalar steplength and the subsequent iterate: \( z^{k+1} = z^k + \lambda d^k \). An Armijo or "back-tracking" line search initially considers \( \lambda = 1 \). If \( \| F(z^k + \lambda d^k) \| \leq \| F(z^k) \| \), the step size \( \lambda \) is adopted, otherwise is multiplied by a positive factor \( \alpha \), \( \alpha < 1 \), and the convergence test is reapplied. This procedure is repeated until either an improvement results or \( \lambda < \lambda_0 \). When \( \lambda_0 = 0 \), a positive step is taken provided that \(^4\)

\[
\frac{d}{d\lambda} \| F(z^k + \lambda d^k) \| < 0.
\]

Convergence theory for this algorithm is quite well developed. See, for example, Ortega and Rheinboldt (1970) or Dennis and Schnabel (1983). The most attractive aspect of the Newton scheme with the backtracking line search is that in the neighborhood of a well-behaved fixed point, \( \lambda = 1 \) is the optimal step length and the rate of convergence can be quadratic. If this method finds a solution, it does so very quickly.

The application of Newton methods to nonlinear complementarity problems involves a modification of the search direction. Here, \( d \) solves a linear complementarity problem (LCP) rather than a linear system of equations. For iteration \( k \), \( d \) solves:

\(^3\)There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse,1992)

\(^4\)\( \alpha \) and \( \lambda_0 \) correspond to user-specified tolerances ‘DMPFAC’ and ‘MINSTP’, respectively
\[
F(z^k) + \nabla F(z^k) d - w + v = 0
\]

\[
\ell \leq d + z^k \leq u, \ w \geq 0, \ v \geq 0
\]

\[
w^T (d + z^k - \ell) = v^T (u - d - z^k) = 0.
\]

Conceptually, we are solving for \(d\), but in practice MILES solves the linear problem in terms of the original variables \(z = z^k + d\):

\[
F(z^k) - \nabla F(z^k) z_k + \nabla F(z^k) z = w - v
\]

\[
\ell \leq z \leq u, \ w \geq 0, \ v \geq 0
\]

\[
w^T (z - \ell) = 0, \ v^T (u - z) = 0.
\]

After computing the solution \(z\), MILES sets \(d^k = z - z^k\).

The linear subproblem incorporates upper and lower bounds on any or all of the variables, assuring that the iterative sequence always remains within the bounds: \(\ell \leq z^k \leq u\). This can be helpful when, as is often the case, \(F()\) is undefined for some \(z \in \mathbb{R}^n\).

Convergence of the Newton algorithm applied to MCP hinges on three questions:

1. Does the linearized problem always have a solution?
2. If the linearized problem has a solution, does Lemke's algorithm find it?
3. Is it possible to show that the computed direction \(d^k\) will provide an "improvement" in the solution?

Only for a limited class of functions \(F()\) can all three questions be answered in the affirmative. For a much larger class of functions, the algorithm converges in practice but convergence is not "provable".

The answer to question 3. depends on the choice of a norm by which an improvement is measured. The introduction of bounds and complementarity conditions makes the calculation of an error index more complicated. In MILES, the deviation associated with a candidate solution \(z, \epsilon(z)\), is based on a measure of the extent to which \(z, w\) and \(v\) violate applicable upper and lower bounds and complementarity conditions.

---

\(^5\)Kaneko (1978) provides some convergence theory for the linearized subproblem
5.25.3.1 Evaluating Convergence

Let $\delta^L_i$ and $\delta^U_i$ be indicator variables for whether $z_i$ is off its lower or upper bound. These are defined as

$$\delta^L_i = \min(1, (z_i - \ell_i)^+) \quad \text{and} \quad \delta^U_i = \min(1, (u_i - z_i)^+).$$

Given $z$, MILES uses the value of $F(z)$ to implicitly define the slack variables $w$ and $v$:

$$w_i = F_i(z)^+, \quad v_i = (-F_i(z))^+.$$

There are two components to the error term associated with index $i$, one corresponding to $z_i$'s violation of upper and lower bounds:

$$\varepsilon^B_i = (z_i - u_i)^+ + (\ell_i - z_i)^+$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon^C_i = \delta^L_i w_i + \delta^U_i v_i.$$

The error assigned to point $z$ is then taken:

$$\varepsilon(z) = \|\varepsilon^B(z) + \varepsilon^C(z)\|_p$$

for a pre-specified value of $p = 1, 2$ or $+\infty$.  

5.25.4 Lemke's Method with Implicit Bounds

A mixed linear complementarity problem has the form:

Given: $M \in \mathbb{R}^{n \times n}$, $q, \ell, u \in \mathbb{R}^n$

Find: $z, w, v \in \mathbb{R}^n$

such that $Mz + q = w - v,$

$\ell \leq z \leq u,$ $w \geq 0,$ $v \geq 0,$

$w^T(z - \ell) = 0,$ $v^T(u - z) = 0.$

In the Newton subproblem at iteration $k$, the LCP data are given by $q = F(z^k) - \nabla F(z^k)z^k$ and $M = \nabla F(z^k)$.

---

6 In the following $x^+ = \max(x, 0)$

7 Parameter $p$ may be selected with input parameter ‘NORM’. The default value for $p$ is $+\infty$. 

5.25 MILES

5.25.4.1 The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

$$\begin{align*}
B x^B + N x^N &= q,
\end{align*}$$

where $x^B \in R^n$, $x^N \in R^{2n}$, and the tableau $[B|N]$ is a conformal "complementary permutation" of $[-M\mid I \mid -I]$. That is, every column $i$ in $B$ must either be the $i$th column of $M, I$ or $-I$, while the corresponding columns $i$ and $i+n$ in $N$ must be the two columns which were not selected for $B$.

To move from the problem defined in terms of $z, w$, and $v$ to the problem defined in terms of $x^B$ and $x^N$, we assign upper and lower bounds for the $x^B$ variables as follows:

$$
x^B_i = \begin{cases} 
\ell_i, & \text{if } x^B_i = z_i \\
0, & \text{if } x^B_i = w_i \text{ or } v_i,
\end{cases}
$$

$$
x^B_i = \begin{cases} 
ut_i, & \text{if } x^B_i = z_i \\
\infty, & \text{if } x^B_i = w_i \text{ or } v_i.
\end{cases}
$$

The values of the non-basic variables $x^N_i$ and $x^N_{i+n}$ are determined by the assignment of $x^B_i$:

$$
x^B_i = \begin{cases} 
z_i \Rightarrow \begin{cases} 
x^N_i = w_i = 0 \\
x^N_{i+n} = v_i = 0
\end{cases} \\
w_i \Rightarrow \begin{cases} 
x^N_i = z_i = \ell_i \\
x^N_{i+n} = v_i = 0
\end{cases} \\
v_i \Rightarrow \begin{cases} 
x^N_i = w_i = 0 \\
x^N_{i+n} = z_i = u_i.
\end{cases}
\end{cases}
$$

In words: if $z_i$ is basic then both $w_i$ and $v_i$ equal zero. If $z_i$ is non-basic at its lower bound, then $w_i$ is possibly non-zero and $v_i$ is non-basic at zero. If $z_i$ is non-basic at its upper bound, then $v_i$ is possibly non-zero and $w_i$ is non-basic at zero.

Conceptually, we could solve the LCP by evaluating $3^n$ linear systems of the form:

$$
x^B = B^{-1} \left( q - N x^N \right).
$$

Lemke’s pivoting algorithm provides a procedure for finding a solution by sequentially evaluating some (hopefully small) subset of these $3^n$ alternative linear systems.
5.25.4.2 Initialization

Let $B^0$ denote the initial basis matrix. The initial values for basic variables are then:

$$\hat{x}^B = (B^0)^{-1}(q - N \tilde{x}^N).$$

If $\hat{x}^B \leq \bar{x}^B \leq \tilde{x}^B$, then the initial basis is feasible and the complementarity problem is solved. Otherwise, MILES introduces an artificial variable $z_0$ and an artificial column $\tilde{h}$. Basic variables are then expressed as follows:

$$x^B = \hat{x}^B - \tilde{h}z_0,$$

where $\tilde{h}$ is the "transformed artificial column" (the untransformed column is $h = B^0\tilde{h}$). The coefficients of $\tilde{h}$ are selected so that:

1. The values of "feasible" basis variables are unaffected by $z_0$: ($x_i^B \leq x_i^B \leq \tilde{x}_i^B \Rightarrow \tilde{h}_i = 0$).
2. The "most infeasible" basic variable ($i = p$) is driven to its upper or lower bound when $z_0 = 1$:

$$\tilde{h}_p = \begin{cases} \frac{\hat{x}_p^B - \bar{x}_p^B}{\bar{x}_p^B - \tilde{x}_p^B}, & \text{if } \bar{x}_p^B > \tilde{x}_p^B \\ \frac{\bar{x}_p^B - \hat{x}_p^B}{\tilde{x}_p^B - \hat{x}_p^B}, & \text{if } \tilde{x}_p^B < \hat{x}_p^B. \end{cases}$$

3. All other infeasible basic variables assume values between their upper and lower bounds when $z_0$ increases to 1:

$$x_i^B = \begin{cases} 1 + x_i^B, & \text{if } x_i^B > -\infty, \quad \hat{x}_i^B = +\infty \\ \frac{x_i^B + \tilde{x}_i^B}{2}, & \text{if } x_i^B > -\infty, \quad \hat{x}_i^B < +\infty \\ \bar{x}_i^B - 1, & \text{if } x_i^B = -\infty, \quad \hat{x}_i^B < +\infty. \end{cases}$$

5.25.4.3 Pivoting Rules

When $z_0$ enters the basis, it assumes a value of unity, and at this point (barring degeneracy), the subsequent pivot sequence is entirely determined. The entering variable in one iteration is determined by the exiting basic variable in the previous iteration. For example, if $z_i$ were in $B^0$ and introducing $z_0$ caused $z_i$ to move onto its lower bound, then the subsequent iteration introduces $w_i$. Conversely, if $w_i$ were in $B^0$ and $z_0$ caused $w_i$ to fall to zero, the subsequent iteration increases $z_i$ from $\ell_i$. Finally, if $v_i$ were in $B^0$ and $z_0$’s introduction caused $v_i$ to fall to zero, the subsequent iteration decreases $z_i$ from $u_i$.

Table 1: Pivot Sequence Rules for Lemke's Algorithm with Implicit Bounds

<table>
<thead>
<tr>
<th>N</th>
<th>Exiting Variable</th>
<th>Entering Variable</th>
<th>Change in Non-basic Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$z_i$ at lower bound</td>
<td>$w_i$ increases from 0</td>
<td>$x_i^N = z_i = \ell_i$</td>
</tr>
<tr>
<td>II</td>
<td>$z_i$ at upper bound</td>
<td>$v_i$ increases from 0</td>
<td>$x_i^N = z_i = u_i$</td>
</tr>
<tr>
<td>III</td>
<td>$w_i$ at 0</td>
<td>$z_i$ increases from $\ell_i$</td>
<td>$x_i^N = x_{i+n}^N = 0$</td>
</tr>
<tr>
<td>IV</td>
<td>$v_i$ at 0</td>
<td>$z_i$ decreases from $u_i$</td>
<td>$x_i^N = x_{i+n}^N = 0$</td>
</tr>
</tbody>
</table>

8In Miles, $B^0$ is chosen using the initially assigned values for $z$. When $z_i \leq \ell_i$, then $x_i^B = w_i$; when $z_i \geq u_i$, then $x_i^B = v_i$; otherwise $x_i^B = z_i$.

9The present version of the code simply sets $B^0 = -I$ and $x^B = w$ when the user-specified basis is singular. A subsequent version of the code will incorporate the algorithm described by Austracher, Lee, and Rutherford [1992] for coping with singularity.
The full set of pivoting rules is displayed in Table 1. One difference between this algorithm and the original Lemke (type III) pivoting scheme (see Lemke (1965), Garcia and Zangwill (1981), or Cottle and Pang (1992)) is that structural variables ($z_i$'s) may enter or exit the basis at their upper bound values. The algorithm, therefore, must distinguish between pivots in which the entering variable increases from a lower bound and those in which the entering variable decreases from an upper bound.

Another difference with the "usual" Lemke pivot procedure is that an entering structural variable may move from one bound to another. When this occurs, the subsequent pivot introduces the corresponding slack variable. For example, if $z_i$ is increased from $\ell_i$ to $u_i$ without driving a basic variable infeasible, then $z_i$ becomes non-basic at $u_i$, and the subsequent pivot introduces $v_i$. This type of pivot may be interpreted as $z_i$ entering and exiting the basis in a single step.\footnote{If all structural variables are subject to finite upper and lower bounds, then no $z_i$ variables may be part of a homogeneous solution adjacent to a secondary ray. This does not imply, however, that secondary rays are impossible when all $z_i$ variables are bounded, as a ray may then be comprised of $w_i$ and $v_i$ variables.}

In theory it is convenient to ignore degeneracy, while in practice degeneracy is unavoidable. The present algorithm does not incorporate a lexicographic scheme to avoid cycling, but it does implement a ratio test procedure which assures that when there is more than one candidate, priority is given to the most stable pivot. The admissible set of pivots is determined on both an absolute pivot tolerance ($\text{ZTOLPV}$) and a relative pivot tolerance ($\text{ZTOLRP}$). No pivot with absolute value smaller than $\min(\text{ZTOLPV}, \text{ZTOLRP} \| V \|)$ is considered, where $\| V \|$ is the norm of the incoming column.

### 5.25.4.4 Termination on a Secondary Ray

Lemke’s algorithm terminates normally when the introduction of a new variable drives $z_0$ to zero. This is the desired outcome, but it does not always happen. The algorithm may be interrupted prematurely when no basic variable "blocks" an incoming variable, a condition known as "termination on a secondary ray". In anticipation of such outcomes, MILES maintains a record of the value of $z_0$ for successive iterations, and it saves basis information associated with the smallest observed value, $z^*_0$. (In Lemke’s algorithm, the pivot sequence is determined without regard for the effect on $z_0$, and the value of the artificial variable may follow an erratic (non-monotone) path from its initial value of one to the final value of zero.)

When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with $z^*_0$ are reinstalled. This basis (augmented with one column from the non-basic triplet to replace $z_0$) serves as $B^0$, and the algorithm is restarted. In some cases this procedure permits the pivot sequence to continue smoothly to a solution, while in other cases may only lead to another secondary ray.

### 5.25.5 The Options File

The standard GAMS options (e.g. iterlim, reslim) can be used to control MILES. For more details, see section Controlling a Solver via GAMS Options.

In addition, MILES-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

The following is a typical MILES options file:

\begin{verbatim}
ITLIMT = 50
CONTOL = 1.0E-8
LUSIZE = 16
\end{verbatim}

In the remainder of this section we describe the MILES options and give their default values.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>contol</td>
<td>convergence tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td></td>
<td>Whenever an iterate is encountered for which $\epsilon(z) &lt; \text{CONTOL}$, the algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>terminates. This corresponds to the GAMS/MINOS parameter &quot;Row tolerance&quot;.</td>
<td></td>
</tr>
<tr>
<td>dens1</td>
<td>LUSOL: density to start searching maxcol columns</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>The density at which the Markowitz strategy should search maxcol columns and no rows. Range:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>dens2</td>
<td>LUSOL: density to start searching 1 column</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>The density at which the Markowitz strategy should search only 1 column or (preferably) use</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a dense LU for all the remaining rows and columns.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>dmpfac</td>
<td>damping factor in backtracking linesearch</td>
<td>0.5</td>
</tr>
<tr>
<td>elmax1</td>
<td>LUSOL: max multiplier allowed in L during factor</td>
<td>10</td>
</tr>
<tr>
<td>elmax2</td>
<td>LUSOL: max multiplier allowed in L during update</td>
<td>10</td>
</tr>
<tr>
<td>factim</td>
<td>basis reinversion time</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Indicates the maximum number of CPU seconds between recalculation of the basis factors.</td>
<td></td>
</tr>
<tr>
<td>invfrq</td>
<td>basis reinversion frequency</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Determines the maximum number of Lemke iterations between recalculation of the basis factors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This corresponds to the GAMS/MINOS parameter &quot;Factorization frequency&quot;.</td>
<td></td>
</tr>
<tr>
<td>invlog</td>
<td>toggle Lemke inversion logging</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A switch which requests LUSOL to generate a report with basis statistics following each</td>
<td></td>
</tr>
<tr>
<td></td>
<td>refactorization.</td>
<td></td>
</tr>
<tr>
<td>itch</td>
<td>iteration refinement frequency</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Indicates the frequency with which the factorization is checked. The number refers to the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>number of basis replacement operations between refinements. This corresponds to the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GAMS/MINOS parameter &quot;Check frequency&quot;.</td>
<td></td>
</tr>
<tr>
<td>iterlim</td>
<td>minor (Lemke) iterations limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td></td>
<td>Can also be set with the GAMS IterLim option, and takes its default from that. This</td>
<td></td>
</tr>
<tr>
<td></td>
<td>corresponds to the GAMS/MINOS parameter &quot;Iterations Limit&quot;.</td>
<td></td>
</tr>
<tr>
<td>itlim</td>
<td>major (Newton) iterations limit</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>An upper bound on the number of Newton iterations. This corresponds to the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GAMS/MINOS parameter &quot;Major iterations&quot;.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 1000}</td>
<td></td>
</tr>
<tr>
<td>lcpdmp</td>
<td>LCP dump (post-scaling)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A switch to generate a printout of the LCP data after scaling.</td>
<td></td>
</tr>
<tr>
<td>lcpech</td>
<td>LCP echo print (pre-scaling)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A switch to generate a printout of the LCP data before scaling, as evaluated.</td>
<td></td>
</tr>
<tr>
<td>levout</td>
<td>set output level</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Sets the level of debug output written to the log and status files. The lowest meaningful</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value is -1 and the highest is 3. This corresponds, roughly, to the GAMS/MINOS parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;Print level&quot;.</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>lprint</td>
<td>LUSOL: print level, controls the amount of logging from the LUSOL routines.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: suppresses output, 1: gives debug output, 2: gives the pivot row and column and the no. of rows and columns involved at each elimination step in lu1fac.</td>
<td></td>
</tr>
<tr>
<td>lusize</td>
<td>LUSOL: multiplier for estimating memory requirements, used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix. Range: {3, ..., \infty}.</td>
<td>5</td>
</tr>
<tr>
<td>maxcol</td>
<td>LUSOL: max cols to search for pivot element, sets the maximum number of columns to search in a Markowitz-type search for the next pivot element. For some of the factorization, the number of rows searched is maxrow = maxcol - 1.</td>
<td>5</td>
</tr>
<tr>
<td>minstp</td>
<td>Minimum step length in backtracking linesearch, norm to use in measuring deviation</td>
<td>0.01</td>
</tr>
<tr>
<td>norm</td>
<td>Defines the vector norm to be used for evaluating (\epsilon(z)). Acceptable values are 1, 2 or 3 which correspond to (p = 1, 2) and (+\infty).</td>
<td>1</td>
</tr>
<tr>
<td>ninmax</td>
<td>Restart limit, sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.</td>
<td>1</td>
</tr>
<tr>
<td>pivlog</td>
<td>Toggle Lemke pivot logging, a switch to generate a status file listing of the Lemke pivot sequence.</td>
<td>0</td>
</tr>
<tr>
<td>plinfy</td>
<td>Infinity used by the solver, the value assigned for &quot;plus infinity&quot; (&quot;+INF&quot; in GAMS notation).</td>
<td>1e20</td>
</tr>
<tr>
<td>scale</td>
<td>Turn on scaling at every iteration, invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch &quot;scale all variables&quot;.</td>
<td>1</td>
</tr>
<tr>
<td>small</td>
<td>LUSOL: absolute zero tolerance, the absolute tolerance for treating reals as zero.</td>
<td>3.0e-13</td>
</tr>
<tr>
<td>uspace</td>
<td>LUSOL: factor limiting waste space in U, the row or column lists are compressed if their length exceeds this value times the length of either file after the last compression. Range: [1, \infty].</td>
<td>3</td>
</tr>
<tr>
<td>utol1</td>
<td>LUSOL: absolute tolerance U diagonal, the absolute tol for flagging small diagonals of U.</td>
<td>3.64e-11</td>
</tr>
<tr>
<td>utol2</td>
<td>LUSOL: relative tolerance U diagonal, the relative tol for flagging small diagonals of U.</td>
<td>3.64e-11</td>
</tr>
<tr>
<td>ztolda</td>
<td>Zero tolerance on matrix coefficients, default: (\text{sqrt(machEps)}).</td>
<td>1.48e-08</td>
</tr>
<tr>
<td>ztolpv</td>
<td>Absolute zero tolerance on pivots, default: (\text{machEps}^{(2/3)}), the absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter &quot;Pivot tolerance&quot; as it applies for nonlinear problems.</td>
<td>3.64e-11</td>
</tr>
<tr>
<td>ztolrp</td>
<td>Relative zero tolerance on pivots, default: (\text{machEps}^{(2/3)}), the relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter &quot;Pivot tolerance&quot; as it applies for nonlinear problems.</td>
<td>3.64e-11</td>
</tr>
<tr>
<td>ztolz0</td>
<td>Absolute tolerance for installing covering vector elements.</td>
<td>1e-6</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ztolze</td>
<td>feasibility tolerance Used in the subproblem solution to determine when any variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter &quot;Feasibility tolerance&quot;.</td>
<td>1e-6</td>
</tr>
</tbody>
</table>

### 5.25.6 Log File Output

The log file is intended for display on the screen in order to permit monitoring progress. Relatively little output is generated.

A sample iteration log is displayed in **Table 2**. This output is from two cases solved in succession. This and subsequent output comes from program `TRNSP.FOR` which calls the MILES library directly. (When MILES is invoked from within GAMS, at most one case is processed at a time.)

The first line of the log output gives the MILES program date and version information. This information is important for bug reports.

The line beginning "Work space..." reports the amount of memory which has been allocated to solve the model - 10K for this example. Thereafter is reported the initial deviation together with the name of the variable associated with the largest imbalance ($\epsilon_B + \epsilon_C$). The next line reports the convergence tolerance.

The lines beginning 0 and 1 are the major iteration reports for those iterations. the number following the iteration number is the current deviation, and the third number is the Armijo step length. The name of the variable complementary to the equation with the largest associated deviation is reported in parenthesis at the end of the line.

Following the final iteration is a summary of iterations, refactorizations, and final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

**Table 2 Sample Iteration Log**

MILES (July 1993) Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email: TOM@GAMS.COM

Work space allocated -- 0.01 Mb

Initial deviation ....... 3.250E+02 P_01
Convergence tolerance .... 1.000E-06

<table>
<thead>
<tr>
<th>Major iterations ....</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemke pivots ..........</td>
<td>10</td>
</tr>
<tr>
<td>Refactorizations ......</td>
<td>2</td>
</tr>
<tr>
<td>Deviation .............</td>
<td>1.137E-13</td>
</tr>
</tbody>
</table>

Solved.
Work space allocated \( \text{--} \) 0.01 Mb

Initial deviation ........ 5.750E+02 W_02
Convergence tolerance ..... 1.000E-06

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Deviation</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.75E+02</td>
<td>1.00E+00 (W_02)</td>
</tr>
<tr>
<td>1</td>
<td>2.51E+01</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>2</td>
<td>4.53E+00</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>3</td>
<td>1.16E+00</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>4</td>
<td>3.05E-01</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>5</td>
<td>8.08E-02</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>6</td>
<td>2.14E-02</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>7</td>
<td>5.68E-03</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>8</td>
<td>1.51E-03</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>9</td>
<td>4.00E-04</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>10</td>
<td>1.06E-04</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>11</td>
<td>2.82E-05</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>12</td>
<td>7.47E-06</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>13</td>
<td>1.98E-06</td>
<td>1.00E+00 (P_01)</td>
</tr>
<tr>
<td>14</td>
<td>5.26E-07</td>
<td>1.00E+00 (P_01)</td>
</tr>
</tbody>
</table>

Major iterations ........ 14
Lemke pivots ............. 23
Refactorizations .......... 15
Deviation ................ 5.262E-07

Solved.

5.25.7 Status File Output

The status file reports more details regarding the solution process than are provided in the log file. Typically, this file is written to disk and examined only if a problem arises. Within GAMS, the status file appears in the listing only following the GAMS statement `OPTION SYSOUT=ON;`.

The level of output to the status file is determined by the options passed to the solver. In the default configuration, the status file receives all information written to the log file together with a detailed listing of all switches and tolerances and a report of basis factorization statistics.

When output levels are increased from their default values using the options file, the status file can receive considerably more output to assist in debugging. Table 3 - Table 6 present a status file generated with `LEVOUT=3` (maximum), `PIVLOG=T`, and `LCPECH=T`.

The status file begins with the same header as the log file. Thereafter is a complete echo-print of the user-supplied option file when one is provided. Following the core allocation report is a full echo-print of control parameters, switches and tolerance as specified for the current run.

Table 4 continues the status file. The iteration-by-iteration report of variable and function values is produced whenever `LEVOUT >= 2`. Table 4 also contains an LCP echo-print. This report has two sections: `$\text{ROWS}$ and `$\text{COLUMNS}$`. The four columns of numbers in the `$\text{ROWS}$` section are the constant vector \((q)\), the current estimate of level values for the associated variables \((z)\), and the lower and upper bounds vectors \((\ell \text{ and } u)\). The letters \(L\) and \(U\) which appear between the \(\text{ROW}\) and \(\text{Z}\) columns are used to identify variables which are non-basic at their lower and upper bounds, respectively. In this example, all upper bounds equal \(+\infty\), so no variables are non-basic at their upper bound.

By convention, only variable (and not equation names) appear in the status file. An equation is identified by the corresponding variable. We therefore see in the `$\text{COLUMNS}$` section of the matrix echo-print, the row names correspond to the names of \(z\) variables. The names assigned to variables \(z_i\), \(w_i\) and \(v_i\) are...
$$z = <\text{name }i>, w = <\text{name }i>, \text{ and } v = <\text{name }i>, \text{ as shown in the $\text{\$COLUMNS}$ section. The nonzeros for } w = <\> \text{ and } v = <\> \text{ variables are not shown because they are assumed to be } -/ + I.$$

The status file output continues on Table 5 where the first half of the table reports output from the matrix scaling procedure, and the second half reports the messages associated with initiation of Lemke’s procedure.

The “lu6chk warning” is a LUSOL report. Thereafter are two factorization reports. Two factorizations are undertaken here because the first basis was singular, so the program installs all the lower bound slacks in place of the matrix defined by the initial values, $z$.

Following the second factorization report, at the bottom of Table 5 is a summary of the initial pivot. “Infeasible in 3 rows.” indicates that $\tilde{h}$ contains 3 nonzero elements. “Maximum infeasibility” reports the largest amount by which a structural variable violates an upper or lower bound. ”Artificial column with 3 elements.” indicates that the vector $h = B^0 \tilde{h}$ contains 3 elements (note that in this case $B^0 = -I$ because the initial basis was singular, hence the equivalence between the number of nonzeros in $\tilde{h}$ and $h$.)

Table 6 displays the final section of the status file. At the top of the table is the Lemke iteration log. The columns are interpreted as follows:

- **ITER** is the iteration index beginning with 0.
- **STATUS** is a statistic representing the efficiency of the Lemke path. Formally, status is the ratio of the minimum number of pivots from $B_0$ to the current basis divided by the actual number of pivots. When the status is 1, Lemke’s algorithm is performing virtually as efficiently as a direct factorization (apart from the overhead of basis factor updates.)
- **Z%** indicates the percentage of columns in the basis are ”structural” ($z$’s).
- **Z0** indicates the value of the artificial variable. Notice that in this example, the artificial variable declines monotonically from its initial value of unity.
- **ERROR** is a column in which the factorization error is reported, when it is computed. For this run, ITCH=30 and hence no factorization errors are computed.
- **INFEAS** is a column in which the magnitude of the infeasibility introduced by the artificial column (defined using the box-norm) is reported. (In MILES the cover vector $h$ contains many different nonzero values, not just 1’s; so there may be a large difference between the magnitude of the artificial variable and the magnitude of the induced infeasibility.
- **PIVOTS** reports the pivot magnitude in both absolute terms (the first number) and relative terms (the second number). The relative pivot size is the ratio of the pivot element to the norm of the incoming column.
- **IN/OUT** report the indices (not names) of the incoming and outgoing columns for every iteration. Notice that Lemke’s iteration log concludes with variable $z_0$ exiting the basis.

The convergence report for iteration 1 is no different from the report written to the log file, and following this is a second report of variable and function values. We see here that a solution has been obtained following a single subproblem. This is because the underlying problem is, in fact, linear.

The status file (for this case) concludes with an iteration summary identical to the log file report and a summary of how much CPU time was employed overall and within various subtasks. (Don’t be alarmed if the sum of the last five numbers does not add up to the first number - some cycles are not monitored precisely.)
NEWTON algorithm control parameters:
  Major iteration limit ..(ITLINT).  25
  Damping factor ..........(DMPFAC).  5.000E-01
  Minimum step length ....(MINSTP).  1.000E-02
  Norm for deviation ......(NORM)...  3
  Convergence tolerance ..(CONTOL).  1.000E-06

LEMKE algorithm control parameters:
  Iteration limit .......(ITERLIM).  1000
  Factorization frequency (INVFRQ).  200
  Feasibility tolerance ..(ZTOLZE).  1.000E-06
  Coefficient tolerance ...(ZTOLDA).  1.483E-08
  Abs. pivot tolerance ... (ZTOLPV).  3.644E-11
  Rel. pivot tolerance ...(ZTOLRP).  3.644E-11
  Cover vector tolerance .(ZTOLZO).  1.000E-06
  Scale every iteration ...(SCALE).  T
  Restart limit ..........(NRSMAX).  1

Output control switches:
  LCP echo print ..........(LCPECH).  F
  LCP dump .................(LCPDMP).  T
  Lemke inversion log ......(INVLOG).  T
  Lemke pivot log ..........(PIVLOG).  T

Initial deviation ........ 3.250E+02  P_01
Convergence tolerance .... 1.000E-06

================================
Convergence Report, Iteration 0
===============================================================
<table>
<thead>
<tr>
<th>ITER</th>
<th>DEVIATION</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.25E+02</td>
<td>1.00E+00 (P_01)</td>
</tr>
</tbody>
</table>

===============================================================

Convergence Report, Iteration 0
### Table 4 Status File with Debugging Output (page 2 of 4)

<table>
<thead>
<tr>
<th>Iteration 0 values.</th>
<th>ROW</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_01_01</td>
<td>L</td>
<td></td>
<td>-7.75000E-01</td>
</tr>
<tr>
<td>X_01_02</td>
<td>L</td>
<td></td>
<td>-8.47000E-01</td>
</tr>
<tr>
<td>X_01_03</td>
<td>L</td>
<td></td>
<td>-8.38000E-01</td>
</tr>
<tr>
<td>X_02_01</td>
<td>L</td>
<td></td>
<td>-7.75000E-01</td>
</tr>
<tr>
<td>X_02_02</td>
<td>L</td>
<td></td>
<td>-8.38000E-01</td>
</tr>
<tr>
<td>X_02_03</td>
<td>L</td>
<td></td>
<td>-8.74000E-01</td>
</tr>
<tr>
<td>W_01</td>
<td>L</td>
<td></td>
<td>3.25000E+02</td>
</tr>
<tr>
<td>W_02</td>
<td>L</td>
<td></td>
<td>5.75000E+02</td>
</tr>
<tr>
<td>P_01</td>
<td>1.0</td>
<td></td>
<td>-3.25000E+02</td>
</tr>
<tr>
<td>P_02</td>
<td>1.0</td>
<td></td>
<td>-3.00000E+02</td>
</tr>
<tr>
<td>P_03</td>
<td>1.0</td>
<td></td>
<td>-2.75000E+02</td>
</tr>
</tbody>
</table>

Function Evaluation, Iteration: 0

$ROWS:

| X_01_01 | -2.25000000E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| X_01_02 | -1.53000000E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| X_01_03 | -1.61999996E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| X_02_01 | -2.25000000E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| X_02_02 | -1.61999996E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| X_02_03 | -1.25999998E-01 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| W_01    | -3.25000000E+02 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+00 |
| W_02    | -5.75000000E+02 | 0.00000000E+00 | 0.00000000E+00 | 1.00000000E+00 |
| P_01    | 3.25000000E+02  | 1.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| P_02    | 3.00000000E+02  | 1.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |
| P_03    | 2.75000000E+02  | 1.00000000E+00 | 0.00000000E+00 | 1.00000000E+20 |

$COLUMNS:

| Z-X_01_01 | W_01  | -1.00000000E+00 |
| Z-X_01_02 | W_01  | -1.00000000E+00 |
| Z-X_01_03 | W_01  | -1.00000000E+00 |
| Z-X_02_01 | W_02  | -1.00000000E+00 |
| Z-X_02_02 | W_02  | -1.00000000E+00 |
| Z-X_02_03 | W_02  | -1.00000000E+00 |
| Z-W_01   | X_01_01 | 1.00000000E+00 |
| Z-W_02   | X_02_01 | 1.00000000E+00 |
| Z-P_01   | X_01_01 | -1.00000000E+00 |
| Z-P_02   | X_01_02 | -1.00000000E+00 |
| Z-P_03   | X_01_03 | -1.00000000E+00 |

...
### Table 5: Status File with Debugging Output (page 3 of 4)

**SCALING LCP DATA**

<table>
<thead>
<tr>
<th></th>
<th>MIN ELEM</th>
<th>MAX ELEM</th>
<th>MAX COL RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFTER 0</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00</td>
</tr>
<tr>
<td>AFTER 1</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00</td>
</tr>
<tr>
<td>AFTER 2</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00</td>
</tr>
<tr>
<td>AFTER 3</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**SCALING RESULTS:**

\[ A(I,J) \leq A(I,J) \times R(I) / C(J) \]

<table>
<thead>
<tr>
<th>ROW</th>
<th>ROW</th>
<th>Z COLUMN</th>
<th>W COLUMN</th>
<th>V COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>4</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>5</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>6</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>7</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>8</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>9</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>11</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

lu6chk warning. The matrix appears to be singular.

- \( nrank = 8 \) rank of U
- \( n - nrank = 3 \) rank deficiency
- \( nsing = 3 \) singularities
- \( jsing = 10 \) last singular column
- \( dumax = 1.00E+00 \) largest triangular diag
- \( dumin = 1.00E+00 \) smallest triangular diag

**LUSOL 5.4 FACTORIZATION STATISTICS**

<table>
<thead>
<tr>
<th>Compressns</th>
<th>0</th>
<th>Merit</th>
<th>0.00</th>
<th>LenL</th>
<th>0</th>
<th>LenU</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase</td>
<td>0.00</td>
<td>M</td>
<td>11</td>
<td>UT</td>
<td>11</td>
<td>D1</td>
<td>0</td>
</tr>
<tr>
<td>Lmax</td>
<td>0.0E+00</td>
<td>Bmax</td>
<td>1.0E+00</td>
<td>Umax</td>
<td>1.0E+00</td>
<td>Umin</td>
<td>1.0E+00</td>
</tr>
<tr>
<td>Growth</td>
<td>1.0E+00</td>
<td>LT</td>
<td>0</td>
<td>BP</td>
<td>0</td>
<td>D2</td>
<td>0</td>
</tr>
</tbody>
</table>

**CONSTRUCTING ARTIFICIAL COLUMN**

--- Infeasible in 3 rows.
--- Maximum infeasibility: 3.250E+02
--- Artificial column with 3 elements.
--- Pivoting in row: 9 to replace column 20
--- Pivot element: -3.250E+02
### Solver Manuals

**ITER STATUS Z% Z0 ERROR INFEAS. ---- PIVOTS ---- IN OUT**

<table>
<thead>
<tr>
<th>ITER</th>
<th>STATUS</th>
<th>Z%</th>
<th>Z0</th>
<th>ERROR</th>
<th>INFEAS.</th>
<th>______</th>
<th>______</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0</td>
<td>1.000</td>
<td>3.E+02</td>
<td>1.E+00</td>
<td>1 Z0 W 9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>9</td>
<td>1.000</td>
<td>1.E+00</td>
<td>1.E+00</td>
<td>2 Z 9 W 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>18</td>
<td>0.997</td>
<td>9.E-01</td>
<td>9.E-01</td>
<td>1 Z 1 W 10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>27</td>
<td>0.997</td>
<td>1.E+00</td>
<td>1.E+00</td>
<td>1 Z 10 W 2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>36</td>
<td>0.996</td>
<td>9.E-01</td>
<td>4.E-01</td>
<td>1 Z 2 W 11</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.00</td>
<td>45</td>
<td>0.996</td>
<td>1.E+00</td>
<td>1.E+00</td>
<td>1 Z 11 W 6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>55</td>
<td>0.479</td>
<td>2.E+00</td>
<td>1.E+00</td>
<td>1 Z 6 W 7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>64</td>
<td>0.479</td>
<td>1.E+00</td>
<td>1.E+00</td>
<td>1 Z 7 W 4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>73</td>
<td>0.000</td>
<td>1.E+00</td>
<td>1.E+00</td>
<td>2 Z 4 W 8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1.00</td>
<td>73</td>
<td>0.000</td>
<td>1.E-03</td>
<td>2.E-03</td>
<td>1 V 8 Z0</td>
<td></td>
</tr>
</tbody>
</table>

**Convergence Report, Iteration 1**

**ITER DEVIATION STEP**

<table>
<thead>
<tr>
<th>ITER</th>
<th>DEVIATION</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.25E+02</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>1</td>
<td>1.14E-13</td>
<td>1.00E+00 (W_02 )</td>
</tr>
</tbody>
</table>

**Iteration 1 values.**

<table>
<thead>
<tr>
<th>ROW</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_01_01</td>
<td>2.50000E+01</td>
<td>-8.32667E-17</td>
</tr>
<tr>
<td>X_01_02</td>
<td>3.00000E+02</td>
<td>-5.55112E-17</td>
</tr>
<tr>
<td>X_01_03</td>
<td>L 0.00000E+00</td>
<td>3.60000E-02</td>
</tr>
<tr>
<td>X_02_01</td>
<td>3.00000E+02</td>
<td>-8.32667E-17</td>
</tr>
<tr>
<td>X_02_02</td>
<td>L 0.00000E+00</td>
<td>8.99999E-03</td>
</tr>
<tr>
<td>X_02_03</td>
<td>2.75000E+02</td>
<td>2.77556E-17</td>
</tr>
<tr>
<td>W_01</td>
<td>1.00000E+00</td>
<td>-1.13687E-13</td>
</tr>
<tr>
<td>W_02</td>
<td>1.00000E+00</td>
<td>1.13687E-13</td>
</tr>
<tr>
<td>P_01</td>
<td>1.22500E+00</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>P_02</td>
<td>1.15300E+00</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>P_03</td>
<td>1.12600E+00</td>
<td>0.00000E+00</td>
</tr>
</tbody>
</table>

Major iterations ........ 1
Lemke pivots ............ 10
Refactorizations ........ 2
Deviation ............... 1.137E-13
Solved.

Total solution time .: 0.6 sec.
Function & Jacobian..: 0.2 sec.
LCP solution ........: 0.2 sec.
Refactorizations ....: 0.1 sec.
FTRAN ..............: 0.0 sec.
Update ..............: 0.1 sec.

**5.25.8 Termination Messages**

**Basis factorization error in INVERT.** An unexpected error code returned by LUSOL. This should normally not occur. Examine the status file for a message from LUSOL 11.

11Within GAMS, insert the line "OPTION SYSOUT=ON;" prior to the solve statement and resubmit the program in order to pass the MILES solver status file through to the listing.
**Failure to converge.** Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

**Inconsistent parameters** $\text{ZTOLZ0}$, $\text{ZTOLZE}$. $\text{ZTOLZ0}$ determines the smallest value loaded into the cover vector $h$, whereas $\text{ZTOLZE}$ is the feasibility tolerance employed in the Harris pivot selection procedure. If $\text{ZTOLZ0} < -\text{ZTOLZE}$, Lemke's algorithm cannot be executed because the initial basis is infeasible.

**Insufficient space for linearization.** Available memory is inadequate for holding the nonzeros in the Jacobian. More memory needs to be allocated. If there is insufficient space for the (Jacobi) matrix, there is far too little memory for holding the LU factors of the same matrix.

**Insufficient space to invert.** More memory needs to be allocated for basis factors. Increase the value of $\text{LUSIZE}$ in the options file, or assign a larger value to `<model>.workspace`.

**Iteration limit exceeded.** This can result from either exceeding the major (Newton) or minor (Lemke) iterations limit. When MILES is invoked from GAMS, the cumulative Lemke iteration limit can be set with the statement `<model>.iterlim = xx>;`. The Newton iteration limit is 100 by default, and it can be modified only through the `ITLIMT` option.

**Resource interrupt.** Elapsed CPU time exceeds options parameter `RESLIM`. To increase this limit, either add `RESLIM = xxx` in the options file or add a GAMS statement `<model>.RESLIM = xxx;`.

**Singular matrix encountered.** Lemke's algorithm has been interrupted due to a singularity arising in the basis factorization, either during a column replacement or during a refactorization. For some reason, a restart is not possible.

**Termination on a secondary ray.** Lemke's algorithm terminated on a secondary ray. For some reason, a restart is not possible.

**Unknown termination status.** The termination status flag has not been set, but the code has interrupted. Look in the status file for a previous message. This termination code should not happen often.

### 5.25.9 References


S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).


Table 7 Transport Model in GAMS/MCP (page 1 of 2)

```gams
TRNSP.GMS

option mcp=miles;

$TITLE LP TRANSPORTATION PROBLEM FORMULATED AS A ECONOMIC EQUILIBRIUM
*=================================================================
* In this file, Dantzig's original transportation model is
* reformulated as a linear complementarity problem. We first
* solve the model with fixed demand and supply quantities, and
* then we incorporate price-responsiveness on both sides of the
* market.
* T.Rutherford 3/91 (revised 5/91)
*=================================================================
* This problem finds a least cost shipping schedule that meets
* requirements at markets and supplies at factories

* References:
* Dantzig, G B., Linear Programming and Extensions
* Princeton University Press, Princeton, New Jersey, 1963,
* Chapter 3-3.
* This formulation is described in detail in Chapter 2
* (A Brooke, D Kendrick and A Meeraus, The Scientific Press,
* Redwood City, California, 1988.

SETS
I canning plants / SEATTLE, SAN-DIEGO /
J markets / NEW-YORK, CHICAGO, TOPEKA / ;

PARAMETERS
A(I) capacity of plant i in cases (when prices are unity)
/ SEATTLE 325
SAN-DIEGO 575 /,
B(J) demand at market j in cases (when prices equal unity)
/ NEW-YORK 325
CHICAGO 300
TOPEKA 275 /,
ESUB(J) Price elasticity of demand (at prices equal to unity)
/ NEW-YORK 1.5
CHICAGO 1.2
TOPEKA 2.0 /;

TABLE D(I,J) distance in thousands of miles

<table>
<thead>
<tr>
<th></th>
<th>NEW-YORK</th>
<th>CHICAGO</th>
<th>TOPEKA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEATTLE</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>SAN-DIEGO</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

SCALAR F freight in dollars per case per thousand miles /90/ ;

PARAMETER C(I,J) transport cost in thousands of dollars per case ;

C(I,J) = F * D(I,J) / 1000 ;

PARAMETER PBAR(J) Reference price at demand node J;
```
POSITIVE VARIABLES
  W(I) shadow price at supply node i,
  P(J) shadow price at demand node j,
  X(I,J) shipment quantities in cases;

EQUATIONS
  SUPPLY(I) supply limit at plant i,
  FXDEMAND(J) fixed demand at market j,
  PRDEMAND(J) price-responsive demand at market j,
  PROFIT(I,J) zero profit conditions;

PROFIT(I,J) .. W(I) + C(I,J) =G= P(J);
SUPPLY(I) .. A(I) =G= SUM(J, X(I,J));
FXDEMAND(J) .. SUM(I, X(I,J)) =G= B(J);
PRDEMAND(J) .. SUM(I, X(I,J)) =G= B(J) * (PBAR(J)/P(J))**ESUB(J);

* Declare models including specification of equation-variable association:
  MODEL FIXEDQTY / PROFIT.X, SUPPLY.W, FXDEMAND.P/ ;
  MODEL EQUILQTY / PROFIT.X, SUPPLY.W, PRDEMAND.P/ ;

* Initial estimate:
  P.L(J) = 1; W.L(I) = 1;

PARAMETER REPORT(*,*,*) Summary report;

SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J); REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

* Calibrate the demand functions:
  PBAR(J) = P.L(J);

* Replicate the fixed demand equilibrium:

SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J); REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "BENCHMARK CALIBRATION", REPORT;

* Compute a counter-factual equilibrium:
  C("SEATTLE","CHICAGO") = 0.5 * C("SEATTLE","CHICAGO");

SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J); REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

* Replicate the fixed demand equilibrium:
SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J);
REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "Reduced Seattle-Chicago transport cost:", REPORT;

5.26 MINOS and QUADMINOS

Bruce A. Murtagh; Graduate School of Management, Macquarie University, Sydney, Australia

Michael A. Saunders, Walter Murray; Department of EESOR, Stanford University, CA

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

5.26.1 Introduction

This document describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver. For a quad-precision MINOS, see quadMINOS.

GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems in a GAMS environment.

GAMS/MINOS is designed to find solutions that are locally optimal. The nonlinear functions in a problem must be smooth (i.e., their first derivatives must exist). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a global optimum. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a starting point that is "sufficiently close", but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

Linearly constrained models are solved with a very efficient and reliable reduced gradient technique that takes advantage of the sparsity of the model. Models with nonlinear constraints are solved with a method that iteratively solves subproblems with linearized constraints and an augmented Lagrangian objective function. This iterative scheme implies that only the final, optimal solution is sure to be feasible w.r.t the nonlinear constraints. This is in contrast to the feasible path method used by some other NLP solvers, e.g., CONOPT. MINOS and CONOPT are very complementary to each other as they employ very different algorithms. See MINOS vs CONOPT for a comparison of the two solvers.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this document.
5.26.2 How to Run a Model with GAMS/MINOS

MINOS is capable of solving many types of models, including LP, NLP, DNLP and QCP. If MINOS is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select MINOS:

\[
\text{option nlp=minos;}
\]

The option statement should appear before the `solve` statement.

To be complete, we mention that the solver can be also specified on the command line, as in:

\[
> \text{gams camcge nlp=minos}
\]

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

5.26.3 Overview of GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

\[
\text{NLP : minimize } \quad \min F(x) + c^T x + d^T y \quad (1)
\]

\[
\text{subject to } \quad f(x) + A_1 y \sim b_1 \quad (2)
\]

\[
A_2 x + A_3 y \sim b_2 \quad (3)
\]

\[
\ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u \quad (4)
\]

where the vectors \( c, d, b_1, b_2, \ell, u \) and the matrices \( A_1, A_2, A_3 \) are constant, \( F(x) \) is a smooth scalar function, and \( f(x) \) is a vector of smooth functions. The \( \sim \) signs mean that individual constraints may be defined using \( \leq, = \) or \( \geq \) corresponding to the GAMS constructs \( =L=, =E= \) and \( =G= \).

The components of \( x \) are called the nonlinear variables, and the components of \( y \) are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let \( m_1 \) and \( n_1 \) denote the number of nonlinear constraints and variables, and let \( m \) and \( n \) denote the total number of (general) constraints and variables. Thus, \( A_3 \) has \( m - m_1 \) rows and \( n - n_1 \) columns. The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of \( \ell \) and \( u \) may be \( -\infty \) or \( +\infty \) respectively, in accordance with the GAMS use of \( -\text{INF} \) and \( +\text{INF} \).

The vectors \( b_1 \) and \( b_2 \) are called the right-hand side, and together are denoted by \( b \).
5.26.3.1 Linear Programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a linear program. Since there is no need to distinguish between linear and nonlinear variables, we use $x$ rather than $y$. GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

\[
\text{LP : minimize } c^T x \\
\text{subject to } Ax + Is = 0 \\
\ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u
\]

where the elements of $x$ are your own GAMS variables, and $s$ is a set of slack variables: one for each general constraint. For computational reasons, the right-hand side $b$ is incorporated into the bounds on $s$.

In the expression $Ax + Is = 0$ we write the identity matrix explicitly if we are concerned with columns of the associated matrix $(A \ I)$. Otherwise we will use the equivalent notation $Ax + s = 0$.

GAMS/MINOS solves linear programs using a reliable implementation of the primal simplex method [41], in which the constraints $Ax + Is = 0$ are partitioned into the form

\[
Bx_B + Nx_N = 0,
\]

where the basis matrix is square and nonsingular. The elements of $x_B$ and $x_N$ are called the basic or nonbasic variables respectively. Together they are a permutation of the vector

\[
\begin{pmatrix} x \\ s \end{pmatrix}.
\]

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equations $Bx_B = Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form.

The simplex method reaches such a solution by performing a sequence of iterations, in which one column of $B$ is replaced by one column of $N$ (and vice versa), until no such interchange can be found that will reduce the value of $c^T x$.

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of $x_B$ lie significantly outside their bounds, we say that the current point is infeasible. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function $c^T x$.

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly between their bounds $\ell_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $\ell_j - \delta < x_j < \ell_j$ or $u_j < x_j < u_j + \delta$ where $\delta$ is a feasibility tolerance (typically $10^{-6}$). In rare cases, even nonbasic variables might lie outside their bounds by as much as $\delta$.

GAMS/MINOS maintains a sparse $LU$ factorization of the basis matrix $B$, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL [82] (see [16] [17] [147] [148] ). The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.
5.26.3.2 Problems with a Nonlinear Objective

When nonlinearities are confined to the term \( F(x) \) in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a reduced-gradient algorithm \[203\] combined with a quasi-Newton algorithm that is described in [139]. In the reduced-gradient method, the constraints \( Ax + Is = 0 \) are partitioned into the form

\[
Bx_B + Sx_S + Nx_N = 0
\]

where \( x_s \) is a set of superbasic variables. At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance \( \delta \), while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be \( s \), the number of columns in \( S \). (The context will always distinguish \( s \) from the vector of slack variables.) At a solution, \( s \) will be no more than \( n_1 \), the number of nonlinear variables. In many practical cases we have found that \( s \) remains reasonably small, say 200 or less, even if \( n_1 \) is large.

In the reduced-gradient algorithm, \( x_s \) is regarded as a set of ”independent variables” or ”free variables” that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of \( B, S \) and \( N \), some of the nonbasic variables are selected to be added to \( S \), and the process is repeated with an increased value of \( s \). At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of \( s \) is reduced by one.

A step of the reduced-gradient method is called a minor iteration. For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix \( Z \) is needed now for descriptive purposes. It takes the form

\[
\begin{pmatrix}
-B^{-1}S \\
I \\
0
\end{pmatrix}
\]

though it is never computed explicitly. Given an \( LU \) factorization of the basis matrix \( B \), it is possible to compute products of the form \( Zq \) and \( Z^Tg \) by solving linear equations involving \( B \) or \( B^T \). This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the \( B, S, N \) partition remains constant. A search direction \( q \) for the superbasic variables is obtained by solving a system of the form

\[
R^T R q = -Z^T g
\]

where \( g \) is a gradient of \( F(x) \), \( Z^T g \) is the reduced gradient, and \( R \) is a dense upper triangular matrix. GAMS computes the gradient vector \( g \) analytically, using automatic differentiation. The matrix \( R \) is updated in various ways in order to approximate the reduced Hessian according to \( R^T R \approx Z^T H Z \) where \( H \) is the matrix of second derivatives of \( F(x) \) (the Hessian).

Once \( q \) is available, the search direction for all variables is defined by \( p = Zq \). A line search is then performed to find an approximate solution to the one-dimensional (w.r.t. \( \alpha \)) problem

\[
\begin{align*}
\text{minimize} & \quad F(x + \alpha p) \\
\text{subject to} & \quad 0 < \alpha < \beta
\end{align*}
\]
where $\beta$ is determined by the bounds on the variables. Another important piece in GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length $\alpha$ (see [80]). The number of nonlinear function evaluations required may be influenced by setting the Linesearch tolerance, as discussed in Section Detailed Description of MINOS Options.

As in a linear programming solver, an equation $B^T\pi = gB$ is solved to obtain the dual variables or shadow prices $\pi$ where $gB$ is the gradient of the objective function associated with basic variables. It follows that $gB - B^T\pi = 0$. The analogous quantity for superbasic variables is the reduced-gradient vector $Z^Tg = gs - s^T\pi$; this should also be zero at an optimal solution. (In practice its components will be of order $r||\pi||$ where $r$ is the optimality tolerance, typically $10^{-6}$, and $||\pi||$ is a measure of the size of the elements of $\pi$.)

### 5.26.3.3 Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a project Lagrangian algorithm, based on a method due to [150], see [140]. This involves a sequence of major iterations, each of which requires the solution of a linearly constrained subproblem. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the $k^\text{th}$ major iteration, let $x_k$ be an estimate of the nonlinear variables, and let $\lambda_k$ be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing $f(x)$ in equation (2) to its linear approximation:

$$f'(x, x_k) = f(x_k) + J(x_k)(x - x_k)$$

or more briefly

$$f' = f_k + J_k(x - x_k)$$

where $J(x_k)$ is the Jacobian matrix evaluated at $x_k$. (The $i$-th row of the Jacobian is the gradient vector of the $i$-th nonlinear constraint function. As with the objective gradient, GAMS calculates the Jacobian using automatic differentiation).

The subproblem to be solved during the $k$-th major iteration is then

$$\text{minimize } F(x) + c^Tx + d^Ty - \lambda_k^T(f - f') + 0.5\rho(f - f')^T(f - f') \tag{5}$$

subject to

$$f' + A_1y \sim b_1 \tag{6}$$
$$A_2x + A_3y \sim b_2 \tag{7}$$
$$\ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u \tag{8}$$

The objective function (5) is called an augmented Lagrangian. The scalar $\rho$ is a penalty parameter, and the term involving $\rho$ is a modified quadratic penalty function.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) – (8). As before, slack variables are introduced and $b_1$ and $b_2$ are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \end{pmatrix}$$

This system will be referred to as $Ax + Is = 0$ as in the linear case. The Jacobian $J_k$ is treated as a sparse matrix, the same as the matrices $A_1$, $A_2$, and $A_3$.

In the output from GAMS/MINOS, the term Feasible subproblem indicates that the linearized constraints have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that feasibility and optimality occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved problems", the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iterations.
5.26.4 Modeling Issues

Formulating nonlinear models requires that the modeler pays attention to some details that play no role when dealing with linear models.

5.26.4.1 Starting Points

The first issue is specifying a starting point. It is advised to specify a good starting point for as many nonlinear variables as possible. The GAMS default of zero is often a very poor choice, making this even more important.

As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points \((x_i, y_i):\)

Example:

\[
\begin{align*}
\text{minimize} & \quad r \\
\text{subject to} & \quad (x_i - a)^2 + (y_i - b)^2 \leq r^2, \quad r \geq 0.
\end{align*}
\]

This problem can be modeled in GAMS as follows.

```gams
set i 'points' /p1*p10/;
parameters
  x(i) 'x coordinates',
  y(i) 'y coordinates';
  * fill with random data
  x(i) = uniform(1,10);
  y(i) = uniform(1,10);
variables
  a 'x coordinate of center of circle'
  b 'y coordinate of center of circle'
  r 'radius';
equations
  e(i) 'points must be inside circle';
e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);
  r.lo = 0;
model m /all/;
option nlp=minos;
solve m using nlp minimizing r;
```

Without help, MINOS will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

\[
\begin{align*}
  x_{\min} &= \min_i x_i, \\
  y_{\min} &= \min_i y_i, \\
  x_{\max} &= \max_i x_i, \\
  y_{\max} &= \max_i y_i,
\end{align*}
\]
then good estimates are

\[ a = \frac{x_{\min} + x_{\max}}{2}, \]
\[ b = \frac{y_{\min} + y_{\max}}{2}, \]
\[ r = \sqrt{\left(a - x_{\min}\right)^2 + \left(b - y_{\min}\right)^2}. \]

Thus we include in our model:

```plaintext
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, y(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set away from zero implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement \( Y.LO = 1; \), the implicit projection of variable levels onto their bounds that occurs when a model is solved will initialize \( Y \) away from zero.

### 5.26.4.2 Bounds

Setting appropriate bounds can be very important to steer the algorithm away from uninteresting areas, and to prevent function evaluation errors from happening.

If your model contains a real power of the form \( x^{**y} \) it is important to add a bound \( x > 0.001 \), as real exponentiation is evaluated in GAMS as \( \exp(y \log(x)) \). In some cases one cannot write a bound directly, e.g. if the equation is \( z = x^{f(y)} \). In that case it is advised to introduce an extra variable and equation:

\[
\begin{align*}
z &= x^\vartheta \\
\vartheta &= f(y) \\
\vartheta &\geq \varepsilon
\end{align*}
\]

(Note that the functions \( \text{SQR}(x) \) and \( \text{POWER}(x, k) \) are integer powers and do not require \( x \) to be positive).

If the model produces function evaluation errors adding bounds is prefered to raising the \( \text{DOMLIM} \) limit.

Bounds in GAMS are specified using \( X.LO(i) = 0.001 \) and \( X.UP(i) = 1000 \).

### 5.26.4.3 Scaling

Although MINOS has some facilities to scale the problem before starting to optimize it, it remains an important task for the modeler to provide a well-scaled model. This is especially the case for nonlinear models. GAMS has special syntax features to specify row and column scales that allow the modeler to keep the equations in a most natural form. For more information consult the GAMS User's Guide.
5.26.4.4 The Objective Function

The first step GAMS/MINOS performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. MINOS however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```gams
obj.. z =e= sum(i, sqr(resid(i)));
model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form NLP (equations (1) − (4)) by saying minimize \( z \) subject to \( z = \sum_i resid_i^2 \) and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize \( \sum_i resid_i^2 \) directly. This can be achieved by a simple reformulation: \( z \) can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable \( z \) is a free continuous variable (no bounds are defined on \( z \)),
- \( z \) appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- \( z \) is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by MINOS. For instance the model [CHEM] from the model library solves in 21 iterations. When we add the bound

```gams
energy.lo = 0;
```

to the objective variable `energy` and thus prevent it from being substituted out, MINOS will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```gams
Variables x,y,z;
Equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
model m /all/;
option nlp=minos;
solve m using nlp minimizing z;
```

will be reformulated as an unconstrained optimization problem

\[
\text{minimize } f(x) = (1 + x)^2.
\]

These additional reformulations can be turned off by using the statement `option reform = 0;` (see Section GAMS Options).
5.26.5 GAMS Options

The standard GAMS options (e.g. iterlim, domlim) can be used to control GAMS/MINOS. For more details, see section Controlling a Solver via GAMS Options. We highlight some of the details of this usage below for cases of special interest.

**iterlim**

Sets the minor iteration limit. MINOS will stop as soon as the number of minor iterations exceeds the iteration limit and report the current solution.

**domlim**

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate \( \sqrt{x} \) for \( x < 0 \). Other examples include taking logs of negative numbers, and evaluating the real power \( x^y \) for \( x < \varepsilon \) (\( x^y \) is evaluated as \( \exp(y \log x) \)). When such an error occurs the count of domain errors is incremented: MINOS will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and MINOS continues. For example, in the case of \( \sqrt{x}, x < 0 \) a zero is passed back for the function value and a large value for the derivative. In many cases MINOS will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression \( \log(x) \) is present in the model, add a statement like \( x.lo = 0.001; \).

**bratio**

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to MINOS so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead crash a new basis. The \textbf{bratio} determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

**workfactor**

By default, GAMS/MINOS computes an estimate of the amount of workspace needed by MINOS, and passes this workspace on to MINOS for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc. In most cases this is sufficient to solve the model. In some rare cases MINOS may need more memory, and the user can provide this by specifying a value of workfactor greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to MINOS for the solve.

**workspace**

The workspace option is deprecated: use the workfactor option instead. The workspace option specifies the amount of memory, in MB, that MINOS will use.

**reform**

This option controls the objective reformulation mechanism described in Section The Objective Function. The default value of 100 will cause GAMS/MINOS to try further substitutions along the diagonal after the objective variable has been removed. Any other value will disable this diagonal procedure.
5.26.6 Summary of MINOS Options

The performance of GAMS/MINOS is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

Note that the option file is not case sensitive. Examples for using the option file can be found at the end of this section. The tables below contain summary information about the MINOS options, default values, and links to more detailed explanations.

5.26.6.1 Output related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug level</td>
<td>Controls amount of debug information written</td>
<td>0</td>
</tr>
<tr>
<td>log frequency</td>
<td>Controls iteration logging to listing file</td>
<td>100</td>
</tr>
<tr>
<td>print level</td>
<td>Controls amount of information printed during optimization</td>
<td>0</td>
</tr>
<tr>
<td>scale print</td>
<td>Print scaling factors</td>
<td></td>
</tr>
<tr>
<td>solution</td>
<td>Prints MINOS solution</td>
<td>NO</td>
</tr>
<tr>
<td>summary frequency</td>
<td>Controls iteration logging to summary (log file)</td>
<td>100</td>
</tr>
</tbody>
</table>

5.26.6.2 Tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash tolerance</td>
<td>Allow crash procedure to ignore small elements in eligible columns</td>
<td>0.1</td>
</tr>
<tr>
<td>feasibility tolerance</td>
<td>Feasibility tolerance for linear constraints</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>linesearch tolerance</td>
<td>Controls accuracy of steplength selected</td>
<td>0.1</td>
</tr>
<tr>
<td>LU density tolerance</td>
<td>When to use dense factorization</td>
<td>0.5</td>
</tr>
<tr>
<td>LU factor tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>100.0</td>
</tr>
<tr>
<td>LU singularity tolerance</td>
<td>Protection against ill-conditioned basis matrices</td>
<td>1.0e-11</td>
</tr>
<tr>
<td>LU update tolerance</td>
<td>Trade-off between stability and sparsity in basis updating</td>
<td>10.0</td>
</tr>
<tr>
<td>optimality tolerance</td>
<td>Reduced gradient optimality check</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>row tolerance</td>
<td>Accuracy requirement for nonlinear rows</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>scale print tolerance</td>
<td>Scale print flag and set tolerance</td>
<td>0.9</td>
</tr>
<tr>
<td>scale tolerance</td>
<td>Scale tolerance</td>
<td>0.9</td>
</tr>
<tr>
<td>subspace tolerance</td>
<td>Determines when nonbasics becomes superbasic</td>
<td>0.5</td>
</tr>
</tbody>
</table>

5.26.6.3 Limits
### 5.26 MINOS and QUADMINOS

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian dimension</td>
<td>Size of Hessian matrix</td>
<td>1</td>
</tr>
<tr>
<td>iterations limit</td>
<td>Minor iteration limit</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>major iterations</td>
<td>Max number of major iterations</td>
<td>50</td>
</tr>
<tr>
<td>minor iterations</td>
<td>Max number of minor iterations between linearizations of nonlinear constraints</td>
<td>40</td>
</tr>
<tr>
<td>superbasics limit</td>
<td>Maximum number of superbasics</td>
<td>1</td>
</tr>
<tr>
<td>unbounded objective value</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e20</td>
</tr>
<tr>
<td>unbounded step size</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e10</td>
</tr>
</tbody>
</table>

### 5.26.6.4 Other algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>check frequency</td>
<td>Controls frequency of linear constraint satisfaction test</td>
<td>60</td>
</tr>
<tr>
<td>completion</td>
<td>Completion level for subproblems (full/partial)</td>
<td>FULL</td>
</tr>
<tr>
<td>crash option</td>
<td>Controls the basis crash algorithm</td>
<td>3</td>
</tr>
<tr>
<td>expand frequency</td>
<td>Setting for anti-cycling mechanism</td>
<td>10000</td>
</tr>
<tr>
<td>factorization frequency</td>
<td>Number of iterations between basis factorizations</td>
<td>100</td>
</tr>
<tr>
<td>lagrangian</td>
<td>Determines form of objection function in the linearized subproblems</td>
<td>YES</td>
</tr>
<tr>
<td>LU complete pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU partial pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU rook pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>major damping parameter</td>
<td>Prevents large relative changes between subproblem solutions</td>
<td>2.0</td>
</tr>
<tr>
<td>minor damping parameter</td>
<td>Limit change in x during linesearch</td>
<td>2.0</td>
</tr>
<tr>
<td>multiple price</td>
<td>Multiple pricing</td>
<td>1</td>
</tr>
<tr>
<td>partial price</td>
<td>Number of segments in partial pricing strategy</td>
<td>10</td>
</tr>
<tr>
<td>penalty parameter</td>
<td>Used in modified augmented Lagrangian</td>
<td>automatic</td>
</tr>
<tr>
<td>radius of convergence</td>
<td>controls final reduction of penalty parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>scale all variables</td>
<td>Synonym to scale option 2</td>
<td></td>
</tr>
<tr>
<td>scale linear variables</td>
<td>Synonym to scale option 1</td>
<td></td>
</tr>
<tr>
<td>scale no</td>
<td>Synonym to scale option 0</td>
<td></td>
</tr>
<tr>
<td>scale nonlinear variables</td>
<td>Synonym to scale option 2</td>
<td></td>
</tr>
<tr>
<td>scale option</td>
<td>Scaling</td>
<td>1</td>
</tr>
<tr>
<td>scale yes</td>
<td>Synonym to scale option 1</td>
<td></td>
</tr>
<tr>
<td>start assigned nonlinear</td>
<td>Starting strategy when there is no basis</td>
<td>SUPERBASIC</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>verify constraint gradients</td>
<td>Synonym to verify level 2</td>
<td></td>
</tr>
<tr>
<td>verify gradients</td>
<td>Synonym to verify level 3</td>
<td></td>
</tr>
<tr>
<td>verify level</td>
<td>Controls verification of gradients</td>
<td>0</td>
</tr>
<tr>
<td>verify no</td>
<td>Synonym to verify level 0</td>
<td></td>
</tr>
<tr>
<td>verify objective gradients</td>
<td>Synonym to verify level 1</td>
<td></td>
</tr>
<tr>
<td>verify yes</td>
<td>Synonym to verify level 3</td>
<td></td>
</tr>
<tr>
<td>weight on linear objective</td>
<td>Composite objective weight</td>
<td>0.0</td>
</tr>
</tbody>
</table>

5.26.6.5 Examples of GAMS/MINOS Option File

The following example illustrates the use of certain options that might be helpful for “difficult” models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

* These options might be relevant for very nonlinear models.
  - Major damping parameter 0.2 * may prevent divergence.
  - Minor damping parameter 0.2 * if there are singularities
    * in the nonlinear functions.
  - Penalty parameter 10.0 * or 100.0 perhaps—a value
    * higher than the default.
  - Scale linear variables * (This is the default.)

Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the cautious defaults are relaxed:

* Suggestions for models with MILDLY nonlinear constraints
  - Completion Full
  - Penalty parameter 0.0 * or 0.1 perhaps—a value
    * smaller than the default.
    * Scale one of the following
  - Scale all variables * if starting point is VERY GOOD.
  - Scale linear variables * if they need it.
  - Scale No * otherwise.

Most of the options should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

5.26.7 Special Notes

5.26.7.1 Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:
• Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix .L).

• Include sensible upper and lower bounds on all variables.

• Specify a *Major damping parameter* that is lower than the default value, if the problem is suspected of being highly nonlinear

• Specify a *Penalty parameter* \( \rho \) that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values \( \lambda_k = 0 \) and \( \rho = 0 \) for all subproblems, by specifying *Lagrangian=No*. However, convergence is much more likely with the default setting, *Lagrangian=Yes*. The initial estimate of the Lagrange multipliers is then \( \lambda_0 = 0 \), but for later subproblems \( \lambda_k \) is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is \( \rho = 100.0/m_1 \) where \( m_1 \) is the number of nonlinear constraints. For later subproblems, \( \rho \) is reduced in stages when it appears that the sequence \( \{x_k, \lambda_k\} \) is converging. In many cases it is safe to specify \( \lambda = 0 \), particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

### 5.26.7.2 Storage

GAMS/MINOS uses one large array of memory for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of physical memory available on a particular machine, and the amount of computation time one is willing to spend.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is \( m \), the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly 100\( m \) doubles, or 800\( m \) bytes for workspace. A further 300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with \( m \) general constraints requires about \( (m + 300) \) K bytes of memory.

Another important quantity is \( n \), the total number of variables in \( x \) and \( y \). The above comments assume that \( n \) is not much larger than \( m \), the number of constraints. A typical ratio for \( n/m \) is 2 or 3.

If there are many nonlinear variables (i.e., if \( n_1 \) is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects \( s \), the number of superbasic variables. Recall that \( s \) is zero for purely linear problems. We know that \( s \) need never be larger than \( n_1 + 1 \). In practice, \( s \) is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix \( R \) has dimension \( s \) and requires about \( s^2/2 \) words of storage. If it seems likely that \( s \) will be very large, some aggregation or reformulation of the problem should be considered.

### 5.26.8 The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section, a sample log file is shown for each case, and the messages that appear are explained.
5.26.8.1 Linear Programs

MINOS uses a standard two-phase simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem TURKPOW is as follows:

GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- turkpow.gms(230) 3 Mb
--- Starting execution: elapsed 0:00:00.009
--- turkpow.gms(202) 4 Mb
--- Generating LP model turkey
--- turkpow.gms(205) 4 Mb
--- 350 rows 949 columns 5,872 non-zeroes
--- Executing MINOS: elapsed 0:00:00.025

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S 5.51 (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.60 Mb

Reading Rows...
Reading Columns...

<table>
<thead>
<tr>
<th>Itn</th>
<th>ninf</th>
<th>sinf</th>
<th>objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3</td>
<td>2.283E-01</td>
<td>-2.51821463E+04</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0.000E+00</td>
<td>2.02819284E+04</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.54107277E+04</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.40211808E+04</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.33804183E+04</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.27082709E+04</td>
</tr>
</tbody>
</table>

EXIT - Optimal Solution found, objective: 12657.77

--- Restarting execution
--- turkpow.gms(205) 0 Mb
--- Reading solution for model turkey
--- turkpow.gms(230) 3 Mb
*** Status: Normal completion

The first line that is written by MINOS is the version string: GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows This line identifies which version of the MINOS libraries and links you are using, and is only to be deciphered by GAMS support personnel.

After some advertisement text we see the amount of work space (i.e. memory) that is allocated: 1.60 Mb. When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on...
statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem loaded into MINOS.

The columns have the following meaning:

**Itn**

Iteration number.

**ninf**

Number of infeasibilities. If nonzero the current iterate is still infeasible.

**sinf**

The sum of the infeasibilities. This number is minimized during Phase I. Once the model is feasible this number is zero.

**objective**

The value of the objective function: \( z = \sum c_i x_i \). In phase II this number is maximized or minimized. In phase I it may move in the wrong direction.

The final line indicates the exit status of MINOS.

### 5.26.8.2 Linearly Constrained NLP's

The log is basically the same as for linear models. The only difference is that not only matrix rows and columns need to be loaded, but also instructions for evaluating functions and gradients.

The log for the problem **WEAPONS** is as follows:

GAMS Rev 235  Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- weapons.gms(77) 3 Mb
--- Starting execution: elapsed 0:00:00.005
--- weapons.gms(66) 4 Mb
--- Generating NLP model war
--- weapons.gms(68) 6 Mb
--- 13 rows 66 columns 156 non-zeros
--- 706 nl-code 65 nl-non-zeros
--- weapons.gms(68) 4 Mb
--- Executing MINOS: elapsed 0:00:00.013

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
MINOS 5.51 (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 0.82 Mb

Reading Rows...
5.26.8.3 NLP’s with Nonlinear Constraints

For models with nonlinear constraints the log is more complicated. The library model [CAMCGE] from the model library is such an example: the log output resulting from running it is shown below.

GAMS Rev 235  Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- camcge.gms(450) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- camcge.gms(441) 4 Mb
--- Generating NLP model camcge
--- camcge.gms(450) 6 Mb
--- 243 rows 280 columns 1,356 non-zeroes
--- 5,524 nl-code 850 nl-non-zeroes
--- camcge.gms(450) 4 Mb
--- Executing MINOS: elapsed 0:00:00.023

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S 5.51  (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.48 Mb

Reading Rows...
Reading Columns...
Reading Instructions...

Major minor step objective Feasible Optimal nsb ncon penalty BSswp
1  2T 0.0E+00 1.91724E+02 1.8E+02 2.0E-01 0  1 1.0E+00 0
2  90 1.0E+00 1.91735E+02 1.5E-03 7.6E+00 0  3 1.0E+00 0
3  0 1.0E+00 1.91735E+02 1.3E-09 5.5E-06 0  4 1.0E+00 0
4  0 1.0E+00 1.91735E+02 1.1E-12 2.8E-13 0  5 1.0E-01 0

EXIT - Optimal Solution found, objective: 191.7346

--- Restarting execution
Two sets of iterations, major and minor, are now reported. A description of the various columns present in this log file follows:

**Major**

A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.

**Minor**

The number of minor iterations performed on the linearized subproblem. If it is a simple number like 90, then the subproblem was solved to optimality. Here, 2T means that the subproblem was terminated. In general the T is not something to worry about. Other possible flags are I and U, which mean that the subproblem was infeasible or unbounded. MINOS may have difficulty if these keep occurring.

**Step**

The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the Major Damping parameter.

**Objective**

The objective function for the original nonlinear program.

**Feasible**

Primal infeasibility, indicating the maximum non-linear constraint violation.

**Optimal**

The maximum dual infeasibility, measured as the maximum departure from complementarity. If we call $d_j$ the reduced cost of variable $x_j$, then the dual infeasibility of $x_j$ is $d_j \times \min\{x_j - \ell_j, 1\}$ or $-d_j \times \min\{u_j - x_j, 1\}$ depending on the sign of $d_j$.

**nsb**

Number of superbasics. If the model is feasible this number cannot exceed the superbasic limit, which may need to be reset to a larger number if the numbers in this column become larger.

**ncon**

The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

**penalty**

The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

**BSswp**

Number of basis swaps: the number of $(B \rightarrow S)$ (i.e. basic vs. superbasic) changes.

Note: The CAMCGE model (like many CGE models or other almost square systems) can better be solved with the MINOS option Start Assigned Nonlinears Basic.
5.26.9 Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. Options not specified will take the default values shown.

check frequency (integer): Controls frequency of linear constraint satisfaction test

Every $i$th iteration after the most recent basis factorization, a numerical test is made to see if the current solution $x$ satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = 0$ where $s$ is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is computed. If the largest component of $r$ is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Range: $\{1, ..., \infty\}$

Default: 60

completion (string): Completion level for subproblems (full/partial)

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion) or to full accuracy (full completion). GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems. Use of partial completion may reduce the work during early major iterations, unless the Minor iterations limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion. An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below $100 \ast (\text{Row tolerance})$, the relative change in $\text{Lambda}_k$ is 0.1 or less, and the previous subproblem was solved to optimality. Full completion tends to give better Lagrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

Default: FULL

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>Solve subproblems to full accuracy</td>
</tr>
<tr>
<td>PARTIAL</td>
<td>Solve subproblems to moderate accuracy</td>
</tr>
</tbody>
</table>

crash option (integer): Controls the basis crash algorithm

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix ($A I$). The value of the parameter $i$ determines which columns of $A$ are eligible. Columns of $I$ are used to fill gaps where necessary. If $i > 0$, three passes are made through the relevant columns of $A$, searching for a basis matrix that is essentially triangular. A column is assigned to pivot on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis). Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality ($=\text{E}=\$) constraints. Pass 3 allows the pivots to be in inequality rows. For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial basis will be a slack basis</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>All columns are eligible</td>
</tr>
<tr>
<td>2</td>
<td>Only linear columns are eligible</td>
</tr>
<tr>
<td>3</td>
<td>Columns appearing nonlinearly in the objective are not eligible</td>
</tr>
<tr>
<td>4</td>
<td>Columns appearing nonlinearly in the constraints are not eligible</td>
</tr>
</tbody>
</table>

**crash tolerance (real):** Allow crash procedure to ignore small elements in eligible columns

The crash tolerance $r$ allows the starting procedure CRASH to ignore certain small nonzeros in each column of $A$. If $a_{max}$ is the largest element in column $j$, other nonzeros $a_{ij}$ in the column are ignored if $|a_{ij}| < a_{max} \times r$. To be meaningful, the parameter $r$ should be in the range $0 \leq r < 1$. When $r > 0.0$ the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of $A$ and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems. For example, suppose the first $m$ columns of $A$ are the matrix shown under LU factor tolerance; i.e., a tridiagonal matrix with entries -1, 4, -1. To help CRASH choose all $m$ columns for the initial basis, we could specify Crash tolerance $r$ for some value of $r > 0.25$.

Range: [0, 1.0]

Default: 0.1

**debug level (integer):** Controls amount of debug information written

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness. Note that you will need to use the GAMS statement `OPTION SYSOUT=on;` to echo the MINOS listing to the GAMS listing file.

- **debug level 0**
  No debug output.
- **debug level 2** (or more)
  Output from `M5SETX` showing the maximum residual after a row check.
- **debug level 40**
  Output from `LU8RPC` (which updates the LU factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
- **debug level 50**
  Output from `LU1MAR` (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
- **debug level 100**
  Output from `M2BFAC` and `M5LOG` listing the basic and superbasic variables and their values at every iteration.

Default: 0

**expand frequency (integer):** Setting for anti-cycling mechanism

This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is $\delta$ and the expand frequency is $k$. Over a period of $k$ iterations, the tolerance actually used by GAMS/MINOS increases from $0.5\delta$ to $\delta$ (in steps $0.5\delta/k$). For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is
at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one
superbasic variable, but otherwise positive steps are enforced. At least every $k$ iterations, a
resetting procedure eliminates any infeasible nonbasic variables. Increasing $k$ helps to reduce
the number of these slightly infeasible nonbasic variables. However, it also diminishes the
freedom to choose a large pivot element (see Pivot tolerance).

Range: $\{1, ..., \infty\}$

Default: 10000

**factorization frequency** *(integer)*: Number of iterations between basis factorizations

At most $i$ basis updates will occur between factorizations of the basis matrix. With linear
programs, basis updates usually occur at every iteration. The default $i$ is reasonable for typical
problems. Higher values up to $i = 200$ (say) may be more efficient on problems that are
extremely sparse and well scaled. When the objective function is nonlinear, fewer basis updates
will occur as an optimum is approached. The number of iterations between basis factorizations
will therefore increase. During these iterations a test is made regularly (according to the Check
frequency) to ensure that the general constraints are satisfied. If necessary the basis will be
re-factorized before the limit of $i$ updates is reached. When the constraints are nonlinear, the
Minor iterations limit will probably preempt $i$.

Range: $\{1, ..., \infty\}$

Default: 100

**feasibility tolerance** *(real)*: Feasibility tolerance for linear constraints

When the constraints are linear, a feasible solution is one in which all variables, including
slacks, satisfy their upper and lower bounds to within the absolute tolerance $r$. (Since
slacks are included, this means that the general linear constraints are also satisfied within $r$.)
GAMS/MINOS attempts to find a feasible solution before optimizing the objective function.
If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let
$SINF$ be the corresponding sum of infeasibilities. If $SINF$ is quite small, it may be appropriate
to raise $r$ by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
If $SINF$ is not small, there may be other points that have a significantly smaller sum of
infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum.
If Scale option $= 1$ or 2, feasibility is defined in terms of the scaled problem (since it is then
more likely to be meaningful). A nonlinear objective function $F(x)$ will be evaluated only at
feasible points. If there are regions where $F(x)$ is undefined, every attempt should be made to
eliminate these regions from the problem. For example, for a function $F(x) = \sqrt{x_1} + \log(x_2)$,
it should be essential to place lower bounds on both variables. If Feasibility tolerance $= 10^{-6}$,
the bounds $x_1 > 10^{-5}$ and $x_2 > 10^{-4}$ might be appropriate. (The log singularity is more serious;
in general, keep variables as far away from singularities as possible.) If the constraints are
nonlinear, the above comments apply to each major iteration. A feasible solution satisfies the
current linearization of the constraints to within the tolerance $r$. The associated subproblem
is said to be feasible. As for the objective function, bounds should be used to keep $x$ more
than $r$ away from singularities in the constraint functions $f(x)$. At the start of major iteration
$k$, the constraint functions $f(x_k)$ are evaluated at a certain point $x_k$. This point always satisfies
the relevant bounds ($l < x_k < u$), but may not satisfy the general linear constraints. During
the associated minor iterations, $F(x)$ and $f(x)$ will be evaluated only at points $x$ that satisfy
the bound and the general linear constraints (as well as the linearized nonlinear constraints).
If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily,
in several stages. Feasibility with respect to the nonlinear constraints themselves is measured
against the Row tolerance (not against $r$). The relevant test is made at the start of a major
iteration.

Default: $1.0e-6$
hessian dimension (integer): Size of Hessian matrix

This specifies that an \( r \times r \) triangular matrix \( R \) is to be available for use by the quasi-Newton algorithm. The matrix \( R \) approximates the reduced Hessian in that \( R^T R \) approximates \( Z^T H Z \). Suppose there are \( s \) superbasic variables at a particular iteration. Whenever possible, \( r \) should be greater than \( s \). If \( r > s \), the first \( s \) columns of \( R \) will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear. If \( r < s \), a matrix of the form

\[
R = \text{diag}(R_r, D)
\]

will be used to approximate the reduced Hessian, where \( R_r \) is an \( r \times r \) upper triangular matrix and \( D \) is a diagonal matrix of order \( s - r \). The rate of convergence will no longer be superlinear (and may be arbitrarily slow). The storage required is of the order \( \text{sqr}(r)/2 \), i.e., quadratic in \( r \). In general, \( r \) should be a slight over-estimate of the final number of superbasic variables, whenever storage permits. It need never be larger than \( n_1 + 1 \), where \( n_1 \) is the number of nonlinear variables. For many problems it can be much smaller than \( n_1 \). If \( \text{Superbasics limit} \) \( s \) is specified, the default value of \( r \) is the same number, \( s \) (and conversely). This is a safeguard to ensure super-linear convergence wherever possible. If neither \( r \) nor \( s \) is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: \( \{1, \ldots, \infty\} \)

Default: 1

iterations limit (integer): Minor iteration limit

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the GAMS ITERLIM specification. If \( i = 0 \), no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

Default: GAMS iterlim

lagrangian (string): Determines form of objection function in the linearized subproblems

This determines the form of the objective function used for the linearized subproblems. The default value \text{yes} \ is highly recommended. The \text{Penalty parameter} \ value is then also relevant. If \text{No} is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not occur.

Default: \text{YES}

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{NO}</td>
<td>Nondefault value (not recommended)</td>
</tr>
<tr>
<td>\text{YES}</td>
<td>Default value (recommended)</td>
</tr>
</tbody>
</table>

linesearch tolerance (real): Controls accuracy of steplength selected

For nonlinear problems, this controls the accuracy with which a steplength \( \alpha \) is located in the one-dimensional problem

\[
\begin{align*}
\text{minimize} & \quad F(x+\alpha p) \\
\text{subject to} & \quad 0 < \alpha \leq \beta 
\end{align*}
\]
A linesearch occurs on most minor iterations for which $x$ is feasible. (If the constraints are nonlinear, the function being minimized is the augmented Lagrangian.) $r$ must be a real value in the range $0.0 < r < 1.0$. The default value $r = 0.1$ requests a moderately accurate search. It should be satisfactory in most cases. If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try $r = 0.01$ or $r = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints. If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try $r = 0.5$ or perhaps $r = 0.9$. (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

Range: $[0, 1.0]$

Default: 0.1

**log frequency (integer):** Controls iteration logging to listing file

In general, one line of the iteration log is printed every $i$th minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions. A value such as $i = 10$, $100$ or larger is suggested for those interested only in the final solution. *Log frequency 0* may be used as shorthand for *Log frequency 99999*. If *Print level* $> 0$, the default value of $i$ is 1. If *Print level* $= 0$, the default value of $i$ is 100. If *Print level* $= 0$ and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

Range: \{1, ..., $\infty$\}

Default: 100

**LU complete pivoting (no value):** LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say $< 2.0$).

**LU density tolerance (real):** When to use dense factorization

The density tolerance is used during LUSOL’s basis factorization $B=LU$. Columns of $L$ and rows of $U$ are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds this tolerance, the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense $LU$ procedure. Raising the tolerance towards 1.0 may give slightly sparser factors, with a slight increase in factorization time.

Range: $[0, 1.0]$

Default: 0.5

**LU factor tolerance (real):** Trade-off between stability and sparsity in basis factorization

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during factorization. The value $r$ specified must satisfy $r \geq 1.0$.

- The default value $r = 100.0$ usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, a larger value may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set $r$ to a value smaller than the default in order to achieve stability.

Range: $[1.0, \infty)$

Default: 100.0

**LU partial pivoting** *(no value)*: LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say $< 2.0$).

**LU rook pivoting** *(no value)*: LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say $< 2.0$).

**LU singularity tolerance** *(real)*: Protection against ill-conditioned basis matrices

When the basis is refactorized, the diagonal elements of $U$ are tested as follows: if $|U_{jj}| < r$ or $|U_{pj}| < r \cdot \max_i |U_{ii}|$, the $j$th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.) In some cases, the Jacobian matrix may converge to values that make the basis very ill-conditioned, causing the optimization to progress very slowly (if at all). Setting $r = 1.0^{-11}$, say, may help cause a judicious change of basis.

Default: $1.0e-11$

**LU update tolerance** *(real)*: Trade-off between stability and sparsity in basis updating

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during updates. The value $r$ specified must satisfy $r > 1.0$.

- The default value $r = 10.0$ usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, $r = 25.0$ (say) may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set $r$ to a value smaller than the default in order to achieve stability.

Range: $[1.0, \infty)$

Default: 10.0

**major damping parameter** *(real)*: Prevents large relative changes between subproblem solutions
The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions \((x_k, \lambda_k)\) and \((x_{k+1}, \lambda_{k+1})\). For example, the default value 2.0 prevents the relative change in either \(x_k\) or \(\lambda_k\) from exceeding 200 percent. It will not be active on well behaved problems. The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus \(x_{k+1}\) and \(\lambda_{k+1}\) are changed to \(x_k + \sigma (x_{k+1} - x_k)\) and \(\lambda_k + \sigma (\lambda_{k+1} - \lambda_k)\) for some step-length \(\sigma < 1\). In the case of nonlinear equations (where the number of constraints is the same as the number of variables) this gives a damped Newton method. This is a very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher Penalty parameter (say 10 or 100 times the default \(\rho\)). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of \(\rho\) is irrelevant.) If the subproblem solutions continue to change violently, try reducing \(r\) to 0.2 or 0.1 (say). For implementation reasons, the shortened step to \(\sigma\) applies to the nonlinear variables \(x\), but not to the linear variables \(y\) or the slack variables \(s\). This may reduce the efficiency of the control.

Default: 2.0

**major iterations** (integer): Max number of major iterations

The maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

Default: 50

**minor damping parameter** (real): Limit change in \(x\) during linesearch

This parameter limits the change in \(x\) during a linesearch. It applies to all nonlinear problems, once a feasible solution or feasible subproblem has been found. A linesearch of the form

\[
\min_{\alpha} F(x + \alpha \cdot p)
\]

is performed over the range \(0 < \alpha < \beta\), where \(\beta\) is the step to the nearest upper or lower bound on \(x\). Normally, the first step length tried is \(\alpha_1 = \min(1, \beta)\). In some cases, such as \(F(x) = ae^{bx}\) or \(F(x) = ax^b\), even a moderate change in the components of \(x\) can lead to floating-point overflow. The parameter \(r\) is therefore used to define a limit

\[
\beta_2 = r(1 + ||x||/||p||)
\]

and the first evaluation of \(F(x)\) is at the potentially smaller steplength \(\alpha_1 = \min(1, \beta, \beta_2)\). Wherever possible, upper and lower bounds on \(x\) should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value \(r = 2.0\) should not affect progress on well behaved problems, but setting \(r = 0.1\) or 0.01 may be helpful when rapidly varying functions are present. A good starting point may be required. An important application is to the class of nonlinear least squares problems. In cases where several local optima exist, specifying a small value for \(r\) may help locate an optimum near the starting point.

Default: 2.0

**minor iterations** (integer): Max number of minor iterations between linearizations of nonlinear constraints

The maximum number of minor iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

Default: 50
The maximum number of feasible minor iterations allowed between successive linearizations of the nonlinear constraints. A moderate value (e.g., \( 20 \leq i \leq 50 \)) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion. The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations (say \( K \)) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify a limit \( > K \), to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. If a good initial subproblem is arbitrarily interrupted by a small limit, the subsequent linearization may be less favorable than the first.

In general it is unsafe to specify a value as small as \( i = 1 \) or \( 2 \). Even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal. The Iteration limit provides an independent limit on the total minor iterations (across all subproblems). If the constraints are linear, only the Iteration limit applies: the minor iterations value is ignored.

Default: 40

**Multiple price (integer):** Multiple pricing

Pricing refers to a scan of the current non-basic variables to determine which, if any, are eligible to become (super)basic. The multiple pricing parameter \( k \) controls the number of entering variables to choose: the \( k \) best non-basic variables are selected for admission to the set of (super)basic variables. The default \( k = 1 \) is best for linear programs, since an optimal solution will have zero superbasic variables. **Warning:** If \( k > 1 \), GAMS/MINOS will use the reduced-gradient method rather than the simplex method, even on purely linear problems. The subsequent iterations do not correspond to the efficient minor iterations carried out by commercial linear programming systems using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving \( k \) dense columns of dimension \( m \), and \( k \) is therefore limited for storage reasons typically to the range \( 2 \leq k \leq 7 \).) GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of \( k \). Larger values of \( k \) are therefore practical, but in general the iterations and time required when \( k > 1 \) are greater than when the simplex method is used (\( k = 1 \)). On large nonlinear problems it may be important to set \( k > 1 \) if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use \( k = 10 \) (say) for early runs, until it seems that the number of superbasics has leveled off. If Multiple price \( k \) is specified, it is also necessary to specify Superbasic limit \( s \) for some \( s > k \).

Range: \{1, ..., ∞\}

Default: 1

**Optimality tolerance (real):** Reduced gradient optimality check

This is used to judge the size of the reduced gradients \( d_j = g_j - pi^T a_j \), where \( g_j \) is the gradient of the objective function corresponding to the \( j^{th} \) variable, \( a_j \) is the associated column of the constraint matrix (or Jacobian), and \( pi \) is the set of dual variables. By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy \( d_j / ||pi|| \geq -r \) or \( d_j / ||pi|| \leq r \) respectively, and if \( d_j / ||pi|| \leq r \) for superbasic variables. The \( ||pi|| \) is a measure of the size of the dual variables. It is included to make the tests independent of a scale factor on the objective function. The quantity actually used is defined by

\[
\sigma = \text{sum}(i, \text{abs}(pi(i))), ||pi|| = \max\{\sigma \text{/sqrt}(m), 1\}
\]

so that only large scale factors are compensated for. As the objective scale decreases, the optimality test tends to become an absolute (instead of a relative) test.

Default: 1.0e-6
**partial price (integer):** Number of segments in partial pricing strategy

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each pricing operation (when a nonbasic variable is selected to become basic or superbasic). When \( i = 1 \), all columns of the constraints matrix \((A \ I)\) are searched. Otherwise, \( A_j \) and \( I \) are partitioned to give \( i \) roughly equal segments \( A_j, I_j \) (\( j = 1 \) to \( i \)). If the previous search was successful on \( A_{j-1}, I_{j-1} \), the next search begins on the segments \( A_j, I_j \). (All subscripts here are modulo \( i \).) If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (Several may be selected if multiple pricing has been specified.) If nothing is found, the search continues on the next segments \( A_{j+1}, I_{j+1} \) and so on. Partial price \( t \) (or \( t/2 \) or \( t/3 \)) may be appropriate for time-stage models having \( t \) time periods.

Range: \( \{1, ..., \infty\} \)

Default: 10

**penalty parameter (real):** Used in modified augmented Lagrangian

This specifies the value of \( \rho \) in the modified augmented Lagrangian. It is used only when \( \text{Lagrangian} = \text{yes} \) (the default setting). For early runs on a problem with unknown characteristics, the default value should be acceptable. If the problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of \( \rho \) may be necessary to ensure convergence, even for convex programs. On the other hand, if \( \rho \) is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify penalty parameter 0.0. When solving a sequence of related problems, initially, use a moderate value for \( \rho \) (such as the default) and a reasonably low Iterations and/or major iterations limit. If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the Major damping parameter.) If there appears to be little progress between major iterations, it may help to reduce the penalty parameter.

Default: automatic

**print level (integer):** Controls amount of information printed during optimization

This varies the amount of information that will be output during optimization. Print level 0 sets the default log and summary frequencies to 100. It is then easy to monitor the progress of the run. Print level 1 (or more) sets the default log and summary frequencies to 1, giving a line of output for every minor iteration. Print level 1 also produces basis statistics, i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized. For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The option value is best thought of as a binary number of the form

\[ \text{Print level JFLXB} \]

where each letter stands for a digit that is either 0 or 1. The quantities referred to are:

- **B** Basis statistics, as mentioned above
- **X** \( x_k \), the nonlinear variables involved in the objective function or the constraints.
- **L** \( \lambda_{k} \), the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if \( \text{Lagrangian}=\text{No} \), since then \( \lambda_{k} = 0 \).)
- **F** \( f(x_k) \), the values of the nonlinear constraint functions.
- **J** \( J(x_k) \), the Jacobian matrix.
To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, 
*Print level 10* sets \( X = 1 \) and the other digits equal to zero; the nonlinear variables will be 
printed each major iteration. If \( J = 1 \), the Jacobian matrix will be output column-wise at the 
start of each major iteration. Column \( j \) will be preceded by the value of the corresponding 
variable \( x_j \) and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence 
if \( J = 1 \), there is no reason to specify \( X = 1 \) unless the objective contains more nonlinear 
variables than the Jacobian.) A typical line of output is

\[
3 \ 1.250000D+01 \ \text{BS} \ 1 \ 1.00000D+00 \ 4 \ 2.00000D+00
\]

which would mean that \( x_3 \) is basic at value 12.5, and the third column of the Jacobian has 
elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually 
different from the GAMS row numbers; see the Solution option.)

Default: 0

**radius of convergence (real):** controls final reduction of penalty parameter \( \rho \)

This determines when the penalty parameter \( \rho \) will be reduced, assuming \( \rho \) was initially 
positive. Both the nonlinear constraint violation (see ROWERR below) and the relative 
change in consecutive Lagrange multiplier estimates must be less than \( r \) at the start of a major 
itration before \( \rho \) is reduced or set to zero. A few major iterations later, full completion 
will be requested if not already set, and the remaining sequence of major iterations should 
converge quadratically to an optimum.

Default: 0.01

**row tolerance (real):** Accuracy requirement for nonlinear rows \( \epsilon \)

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The 
default value is usually small enough, since model data is often specified to about this accuracy. 
Let \( \text{ROWERR} \) be the maximum component of the residual vector \( f(x) + A_1 y - b_1 \), normalized 
by the size of the solution. Thus

\[
\text{ROWERR} = \frac{||f(x) + A_1 y - b_1||_\infty}{(1 + \text{XNORM})}
\]

where \( \text{XNORM} \) is a measure of the size of the current solution \( (x, y) \). The solution is considered 
 to be feasible if \( \text{ROWERR} <= r \). If the problem functions involve data that is known to be of 
low accuracy, a larger Row tolerance may be appropriate.

Default: 1.0e-6

**scale all variables (no value):** Synonym to scale option 2 \( \epsilon \)

**scale linear variables (no value):** Synonym to scale option 1 \( \epsilon \)

**scale no (no value):** Synonym to scale option 0 \( \epsilon \)

**scale nonlinear variables (no value):** Synonym to scale option 2 \( \epsilon \)

**scale option (integer):** Scaling \( \epsilon \)

*Scale Yes* sets the default. (*Caution:* If all variables are nonlinear, *Scale Yes* unexpectedly does 
nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent 
to *Scale Option 0*). If nonlinear constraints are present, *Scale option 1* or *0* should generally 
be tried at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should 
therefore be used only if (a) a good starting point is provided, and (b) the problem is not 
highly nonlinear.

Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No scaling&lt;br&gt; If storage is at a premium, this option should be used.</td>
</tr>
<tr>
<td>1</td>
<td>Scale linear variables&lt;br&gt; Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see [5]). This will sometimes improve the performance of the solution procedures. Scale linear variables is an equivalent option.</td>
</tr>
<tr>
<td>2</td>
<td>Scale linear + nonlinear variables&lt;br&gt; All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side $b$ or the solution $x$ is large. This takes into account columns of $(A I)$ that are fixed or have positive lower bounds or negative upper bounds. Scale nonlinear variables or Scale all variables are equivalent options.</td>
</tr>
</tbody>
</table>

**scale print** *(no value)*: Print scaling factors ←

This causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\hat{a}_{ij} = a_{ij}c(j)/r(i)$. The scaled bounds on the variables and slacks are $\hat{b}_j = b_j/c(j)$ and $\hat{u}_j = u_j/c(j)$, where $c(j) = r(j - n)$ if $j > n$. If a Scale option has not already been specified, **Scale print** sets the default scaling.

**scale print tolerance** *(real)*: Scale print flag and set tolerance ←

See **Scale Tolerance**. This option also turns on printing of the scale factors.

Range: [0, 1.0]  
Default: 0.9

**scale tolerance** *(real)*: Scale tolerance ←

All forms except **Scale option** may specify a tolerance $r$ where $0 < r < 1$ (for example: **Scale Print Tolerance** = 0.99). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_{ij} = \max_i |a_{ij}|/\min_i |a_{ij}| \quad (a_{ij} \neq 0)$$

If $\max_j \rho_{ij}$ is less than $r$ times its previous value, another scaling pass is performed to adjust the row and column scales. Raising $r$ from 0.9 to 0.99 (say) usually increases the number of scaling passes through $A$. At most 10 passes are made. If a **Scale option** has not already been specified, **Scale tolerance** sets the default scaling.

Range: [0, 1.0]  
Default: 0.9

**scale yes** *(no value)*: Synonym to scale option 1 ←

**solution** *(string)*: Prints MINOS solution ←

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS column numbers, and those at the right of each line in the rows section are GAMS/MINOS slacks.) The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

Default: NO
**start assigned nonlinear**s *(string)*: Starting strategy when there is no basis

This option affects the starting strategy when there is no basis (i.e., for the first solve or when the GAMS statement `option bratio = 1` is used to reject an existing basis.) This option applies to all nonlinear variables that have been assigned nondefault initial values and are strictly between their bounds. Free variables at their default value of zero are excluded. Let \( K \) denote the number of such assigned nonlinear variables.

Default: **SUPERBASIC**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPERBASIC</td>
<td>Default Specify <em>superbasic</em> for highly nonlinear models, as long as ( K ) is not too large (say ( K &lt; 100 )) and the initial values are good.</td>
</tr>
<tr>
<td>BASIC</td>
<td>Good for square systems Specify <em>basic</em> for models that are essentially square (i.e., if there are about as many general constraints as variables).</td>
</tr>
<tr>
<td>NONBASIC</td>
<td>Specify <em>nonbasic</em> if ( K ) is large.</td>
</tr>
<tr>
<td>ELIGIBLE FOR CRASH</td>
<td>Specify <em>Eligible for Crash</em> for linear or nearly linear models. The nonlinear variables will be treated in the manner described under <em>Crash</em> option.</td>
</tr>
</tbody>
</table>

**subspace tolerance** *(real)*: Determines when nonbasics becomes superbasic

This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3). The parameter \( r \) must be a real number in the range \( 0 < r <= 1 \). When a nonbasic variable \( x_j \) is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be \( \|Z^T g_0\| \). (In fact, the norm will be \( |d_j| \), the size of the reduced gradient for the new superbasic variable \( x_j \).) Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies \( \|Z^T g_0\| < r \|Z^T g_0\| \) is the size of the largest reduced-gradient component among the superbasic variables.) A smaller value of \( r \) is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as \( r = 0.9 \) may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution. Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of \( r \).

Range: \([0, 1.0]\)

Default: 0.5

**summary frequency** *(integer)*: Controls iteration logging to summary (log file)

A brief form of the iteration log is output to the MINOS summary file (i.e. the GAMS log file). In general, one line is output every \( r \)th minor iteration. In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated. The summary frequency controls summary output in the same way as the log frequency controls output to the print file. A value such as
Summary Frequency = 10 or 100 is often adequate to determine if the solve is making progress. If Print level = 0, the default value of Summary Frequency is 100. If Print level > 0, the default value of Summary Frequency is 1. If Print level = 0 and the constraints are nonlinear, the Summary Frequency is ignored. Instead, one line is printed at the beginning of each major iteration.

Range: \{1, \ldots, \infty\}

Default: 100

superbasics limit (integer): Maximum number of superbasics

This places a limit on the storage allocated for superbasic variables. Ideally, the parameter $i$ should be set slightly larger than the number of degrees of freedom expected at an optimal solution. For linear problems, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is not more than $m$, the number of general constraints.) The default value of $i$ is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the number of independent variables. Normally, $i$ need not be greater than $n_1 + 1$, where $n_1$ is the number of nonlinear variables. For many problems, $i$ may be considerably smaller than $n_1$. This will save storage if $n_1$ is very large. This parameter also sets the Hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: \{1, \ldots, \infty\}

Default: 1

unbounded objective value (real): Determines when a problem is called unbounded

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\min_{\alpha} F(x + \alpha p)$$

If $|F|$ exceeds the parameter $r$ or if $\alpha$ exceeds the unbounded stepsize, iterations are terminated with the exit message PROBLEM IS UNBOUNDED (OR BADLY SCALED). If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against $r$ can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the Minor damping parameter.

Default: 1.0e20

unbounded step size (real): Determines when a problem is called unbounded

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\min_{\alpha} F(x + \alpha p)$$

If $\alpha$ exceeds the parameter $r$ or if $|F|$ exceeds the unbounded objective value, iterations are terminated with the exit message PROBLEM IS UNBOUNDED (OR BADLY SCALED). If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against $r$ can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the Minor damping parameter.

Default: 1.0e10
**verify constraint gradients** *(no value)*: Synonym to verify level 2 ←

**verify gradients** *(no value)*: Synonym to verify level 3 ←

**verify level** *(integer)*: Controls verification of gradients ←

This option controls the finite-difference check performed by MINOS on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as correct.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
</table>
| 0     | Cheap test  
Only a cheap test is performed, requiring three evaluations of the nonlinear objective and two evaluations of the nonlinear constraints. **Verify No** is an equivalent option. |
| 1     | Check objective  
A more reliable check is made on each component of the objective gradient. **Verify objective gradients** is an equivalent option. |
| 2     | Check Jacobian  
A check is made on each column of the Jacobian matrix associated with the nonlinear constraints. **Verify constraint gradients** is an equivalent option. |
| 3     | Check objective and Jacobian  
A detailed check is made on both the objective and the Jacobian. **Verify, Verify gradients**, and **Verify Yes** are equivalent options. |
| -1    | No check |

**verify no** *(no value)*: Synonym to verify level 0 ←

**verify objective gradients** *(no value)*: Synonym to verify level 1 ←

**verify yes** *(no value)*: Synonym to verify level 3 ←

**weight on linear objective** *(real)*: Composite objective weight ←

This option controls the so-called composite objective technique. If the first solution obtained is infeasible, and if the objective function contains linear terms, and the objective weight \( w \) is positive, this technique is used. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear portion of the objective. At each infeasible iteration, the objective function is defined to be

\[
\text{minimize}_x \sigma w(c^T x) + \text{(sum of infeasibilities)}
\]

where \( \sigma = 1 \) for minimization and \( \sigma = -1 \) for maximization and \( c \) is the linear portion of the objective. If an optimal solution is reached while still infeasible, \( w \) is reduced by a factor of 10. This helps to allow for the possibility that the initial \( w \) is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending towards zero. The effect of \( w \) is disabled after five such reductions, or if a feasible solution is obtained. This option is intended mainly for linear programs. It is unlikely to be helpful if the objective function is nonlinear.

Default: 0.0
5.26.10 Exit Conditions

This section discusses the exit codes printed by MINOS at the end of the optimization run.

EXIT – Optimal solution found

This is the message we all hope to see! It is certainly preferable to every other message. Of course it is quite possible that there are model formulation errors, which will (hopefully) lead to unexpected objective values and solutions. The reported optimum may be a local, and other much better optima may exist.

EXIT – The problem is infeasible

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables (the bounds on the slack variables correspond to the GAMS constraints). The message tells us that among all the points satisfying the general constraints \( Ax + s = 0 \), there is apparently no point that satisfies the bounds on \( x \) and \( s \). Violations as small as the FEASIBILITY TOLERANCE are ignored, but at least one component of \( x \) or \( s \) violates a bound by more than the tolerance.

Note: Although the objective function is the sum of the infeasibilities, this sum will usually not have been minimized when MINOS recognizes the situation and exits. There may exist other points that have significantly lower sum of infeasibilities.

When nonlinear constraints are present, infeasibility is much harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation MINOS may relax the bounds on the slacks associated with nonlinear rows. This perturbation is allowed a fixed number of times. Normally a feasible point will be obtained relative to the perturbed constraints, and optimization can continue on the subproblem. However, if several consecutive subproblems require such perturbation, the problem is terminated and declared INFEASIBLE. Clearly this is an ad-hoc procedure. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT – The problem is unbounded (or badly scaled)

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased by an arbitrary amount without causing a basic variable to violate a bound. A simple way to diagnose such a model is to add an appropriate bound on the objective variable.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the SCALE option.

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the UNBOUNDED parameter), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

EXIT – User Interrupt

This exit code is a result of interrupting the optimization process by hitting Ctrl-C. Inside the IDE this is accomplished by hitting the Interrupt button. The solver will finish its current iteration, and return the current solution. This solution can be still intermediate infeasible or intermediate non-optimal.

EXIT – Too many iterations
The iteration limit was hit. Either the ITERLIM, or in some cases the ITERATIONS LIMIT or MAJOR ITERATION LIMIT was too small to solve the problem. In most cases increasing the GAMS ITERLIM option will resolve the problem. In other cases you will need to create a MINOS option file and set a MAJOR ITERATION LIMIT. The listing file will give more information regarding what limit was hit.

The GAMS iteration limit is displayed in the listing file under the section SOLVE SUMMARY. If the ITERLIM was hit, the message will look like:

```
ITERATION COUNT, LIMIT  10001  10000
```

**EXIT – Resource Interrupt**

The solver hit the RESLIM resource limit, which is a time limit. It returned the solution at that time, which may be still intermediate infeasible or intermediate non-optimal.

The GAMS resource limit is displayed in the listing file under the section SOLVE SUMMARY. If the GAMS RESLIM was hit, the message will look like:

```
RESOURCE USAGE, LIMIT  1001.570  1000.000
```

**EXIT – The objective has not changed for many iterations**

This is an emergency measure for the rare occasions when the solution procedure appears to be *cycling*. Suppose that a zero step is taken for several consecutive iterations, with a basis change occurring each time. It is theoretically possible for the set of basic variables to become the same as they were one or more iterations earlier. The same sequence of iterations would then occur *ad infinitum*.

**EXIT – The Superbasics Limit is too small**

The problem appears to be more non-linear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and an increase in the number of superbasics is needed. You can use the option SUPERBASICS LIMIT to increase the limit. See also option HESSIAN DIMENSION.

**EXIT – Constraint and objective function could not be calculated**

The function or gradient could not be evaluated. For example, this can occur when MINOS attempts to take a log or a square root of a negative number, when evaluating the expression $x^y$ with $x \leq 0$, or when evaluating $\exp(x)$ for large $x$ and the result is too large to store. The listing file will contain details about where and why evaluation errors occur. To fix this problem, add bounds so that all functions can be properly evaluated. E.g. if you have an expression $x^y$, add a lower bound $x.LO=0.001$ to your model.

In many cases the algorithm can recover from function evaluation errors, for instance if they happen in the line search while evaluating trial points. The message above appears in cases where the algorithm cannot recover, and requires a reliable function or gradient evaluation.

**EXIT – Function evaluation error limit**
The limit of allowed function evaluation errors $\text{DOMLIM}$ has been exceeded.

Function evaluation errors occur when MINOS attempts to evaluate the objective and/or constraints at points where these functions or their derivatives are not defined or where overflows occur. Some examples are given above. The listing file contains details about these errors.

The quick and dirty way to solve this is to increase the GAMS $\text{DOMLIM}$ setting, but in general it is better to add bounds. E.g. if you have an expression $x^y$, then add a bound $x.LO=0.001$ to your model.

EXIT – The current point can not be improved

The line search failed. This can happen if the model is very nonlinear or if the functions are nonsmooth (using a $\text{DNLP}$ model type).

If the model is non-smooth, consider a smooth approximation. It may be useful to check the scaling of the model and think more carefully about choosing a good starting point. Sometimes it can help to restart the model with full scaling turned on:

```gams
option nlp=minos;
solve m minimizing z using nlp; // this one gives "current point cannot be improved"
file fopt /minos.opt/; // write option file
putclose fopt "scale all variables"/
m.optfile=1;
solve m minimizing z using nlp; // solve with "scale all variables"
```

EXIT – Numerical error in trying to satisfy the linear constraints (or the linearized constraints)
The basis is very ill-conditioned.

This is often a scaling problem. Try the full scaling option $\text{scale all variables}$ or, better yet, rescale the model in GAMS via the $\text{.scale}$ suffix or by choosing more appropriate units for variables and RHS values.

EXIT – Not enough storage to solve the model

The amount of workspace allocated for MINOS to solve the model is insufficient. Consider increasing the GAMS option $\text{workfactor}$ to increase the workspace allocated for MINOS to use. The listing file and log file (screen) will contain information about the current workspace allocation. Increasing the workfactor by 50% is a reasonable strategy.

EXIT – Systems error

This is a catch all return for other serious problems. Check the listing file for more messages. If needed rerun the model with $\text{OPTION SYSOUT=ON;}$.

5.27 MOSEK

MOSEK ApS, C/O Symbion Science Park, Fruebjergvej 3, Box 16, 2100 Copenhagen Ø, Denmark
5.27 MOSEK

5.27.1 Introduction

MOSEK is a software package for the solution of linear, mixed-integer linear, quadratic, mixed-integer quadratic, quadratically constraint, conic, and semidefinite mathematical optimization problems. MOSEK is particularly well suited for solving large-scale linear, convex quadratically constraint, and conic programs using an extremely efficient interior point algorithm.

These problem classes can be solved using an appropriate optimizer built into MOSEK. All the optimizers available in MOSEK are built for the solution of large-scale sparse problems. Current optimizers include:

- Interior-point optimizers for continuous and conic problems
- Simplex optimizer for linear problems
- Branch-and-cut optimizer for mixed-integer linear, quadratic, and conic problems

5.27.1.1 Licensing

Licensing of GAMS/MOSEK is similar to other GAMS solvers. MOSEK is licensed in three different ways:

- **GAMS/MOSEK:**
  All model types.

- **GAMS/MOSEK Solver Link:**
  Users must have a separate, licensed MOSEK system. For users who wish to use MOSEK within GAMS and also in other environments.

- **GAMS/OsiMosek:**
  Finally, a bare-bone interface to the LP and MIP solver of Mosek is available under the name OSIMOSEK. It comes free of charge with any GAMS system.

For information regarding MOSEK standalone or interfacing MOSEK with other applications contact sales@mosek.com.

5.27.1.2 Solving Problems in Parallel

MOSEK can exploit multiple CPUs (or a CPU with multiple cores) to solve an optimization problem when using the interior-point or the mixed-integer optimizers.

This implies that whenever the MOSEK interior-point optimizer should solve an optimization problem, then it will try to divide the work so each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit. Unfortunately, it is not always easy to divide the work. Also some of the coordination work must occur in sequential. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e., in less than 60 seconds, then there is no advantage in using the parallel option.

The parameter `MSK_IPAR_NUM_THREADS` sets the number of threads (and therefore the number of CPU's) that the optimizer will use.
5.27.1.3 Infeasible/Unbounded Models

5.27.1.3.1 Farkas Certificates MOSEK determines if either the primal or the dual problem is infeasible by means of a Farkas certificate. In such a case MOSEK returns a certificate indicating primal or dual infeasibility.

The primal infeasibility certificate indicates a primal infeasible model. For a minimization problem

\[
\begin{align*}
\text{minimize} & \quad \langle c, x \rangle \\
\text{subject to} & \quad Ax = b, \\
& \quad x \geq 0,
\end{align*}
\]

the primal infeasibility certificate is the solution \( y \) satisfying \( A'y \leq 0 \) and \( \langle b, y \rangle > 0 \).

A primal infeasibility certificate is reported in the marginal records of the variables and equations. As no primal solution is available in this case, the level values for variables and equations and the objective function value are set to 0 (setting them to NA would be more appropriate, but GAMS does not support this well). At the moment, primal infeasibility certificate are not available for conic programs.

Since GAMS reports all model statuses in the primal space, the notion of dual infeasibility does not exist and GAMS reports a status of unboundedness, which assumes that the primal problem is feasible. Although GAMS reports the primal as unbounded, there is the possibility that both the primal and dual problem are infeasible. To check if this is the case, the user can set appropriate lower and upper bounds on the objective variable, using the \((\text{variable}).\text{LO}\) and \((\text{variable}).\text{UP}\) suffixes and resolve.

The dual infeasibility certificate is reported in the level values for the variables. As no dual solution exists, the marginal values for both variables and equations are set to NA.

For more details on primal and dual infeasibility certificates see the MOSEK Modeling Cookbook.

5.27.1.3.2 Infeasibility Report MOSEK has some facilities for diagnosing the cause of a primal or dual infeasibility. They can be turned on using the parameter setting \texttt{MSK_IPAR_INFEAS_REPORT.AUTO}. This causes MOSEK to print a report about an infeasible subset of the constraints, when an infeasibility is encountered. Moreover, the parameter \texttt{MSK_IPAR_INFEAS_REPORT.LEVEL} controls the amount of information presented in the infeasibility report. We will use the \texttt{TRANSPORT} example from the GAMS Model Library with increased demand \((b(j) \leftarrow 1.6b(j))\) to make the model infeasible. MOSEK produces the following infeasibility report:

**MOSEK PRIMAL INFEASIBILITY REPORT.**

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Dual lower</th>
<th>Dual upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>supply(seattle)</td>
<td>none</td>
<td>3.500000e+002</td>
<td>0.000000e+000</td>
<td>1.000000e+000</td>
</tr>
<tr>
<td>2</td>
<td>supply(san-diego)</td>
<td>none</td>
<td>6.000000e+002</td>
<td>0.000000e+000</td>
<td>1.000000e+000</td>
</tr>
<tr>
<td>3</td>
<td>demand(new-york)</td>
<td>5.200000e+002</td>
<td>none</td>
<td>1.000000e+000</td>
<td>0.000000e+000</td>
</tr>
<tr>
<td>4</td>
<td>demand(chicago)</td>
<td>4.800000e+002</td>
<td>none</td>
<td>1.000000e+000</td>
<td>0.000000e+000</td>
</tr>
<tr>
<td>5</td>
<td>demand(topeka)</td>
<td>4.400000e+002</td>
<td>none</td>
<td>1.000000e+000</td>
<td>0.000000e+000</td>
</tr>
</tbody>
</table>

The following bound constraints are involved in the infeasibility.

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Dual lower</th>
<th>Dual upper</th>
</tr>
</thead>
</table>

The report indicates which constraints and bounds are causing the infeasibility. In this case, the constraints causing infeasibility are \texttt{supply} and \texttt{demand}. The values in the columns \texttt{Dual lower} and \texttt{Dual upper} are also useful, because if the dual lower value is different from zero for a constraint, then it implies that the lower bound on the constraint is important for the infeasibility. Similarly, if the dual upper value is different from zero on a constraint, then this implies the upper bound on the constraint is important for infeasibility.
5.27 MOSEK

5.27.1.4 Conic Programming

MOSEK is well suited for solving generalized linear programs involving certain conic constraints.

For an overview of quadratic conic programming and how these conic constraints are implemented in GAMS, see the Section Conic Programming in the GAMS User’s Guide. Note, that for Mosek no variable can appear in more than one conic constraint.

Additionally, the primal power cone, defined as

\[ x_0^\alpha x_1^{1-\alpha} \geq \sqrt{n \sum_{i=2}^{n} x_i^2}, \quad x_0, x_1 \geq 0, \]

with \( \alpha \in (0, 1) \), and the primal exponential cone, defined as

\[ x_0 \geq x_1 \exp(x_2/x_1), \quad x_0, x_1 \geq 0, \]

are available.

Since GAMS does not offer capabilities to directly restrict a variable to one of these cones, the GAMS/MOSEK link tries to detect the above algebra from a general nonlinear equation. For example, the following GAMS code should work with MOSEK:

Set i / 0*10 /;
Variable x(i);
Equations e1, e2;
Scalar alpha;
e1.. x('0')**alpha * x('1')**(1-alpha) =G= sqrt(sum(i$(ord(i)>2), sqr(x(i))));
e2.. x('0') =G= x('1') * exp(x('2') / x('1'));
x.lo('0') = 0;
x.lo('1') = 0;

See also testlib models powercone1, powercone2, and expcone1.

5.27.2 Solver Options

MOSEK works like other GAMS solvers, and many options can be set in the GAMS model (see GAMS Options). The most relevant GAMS options are reslim, nodlim, iterlim (iteration limit for simplex and interior point algorithms), optca, optcr, and optfile. A description of all available GAMS options can be found in GAMS Options and Solver related options.

We remark that MOSEK contains many complex solver options, many of which require a deep understanding of the algorithms used. For information on how to use a GAMS/Mosek options file, see The Solver Option File. For example, an option file

\[
\begin{align*}
&\text{MSK_IPAR_INTPNT_MAX_ITERATIONS} & 20 \\
&\text{MSK_IPAR_INTPNT_SCALING} & \text{MSK_SCALING_NONE}
\end{align*}
\]

limits the number of interior-point iterations to 20 and disables scaling.

In the following, we summarize the MOSEK options that are available through the GAMS/MOSEK interface. For details, see Detailed Descriptions of MOSEK Options.

5.27.2.1 General
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_OPTIMIZER_MAX_TIME</td>
<td>Maximum amount of time the optimizer is allowed to spend on the optimization.</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>MSK_DPAR_SEMIDEFINITE_TOL_APPROX</td>
<td>Tolerance to define a matrix to be positive semidefinite.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_IPAR_AUTO_SORT_A_BEFORE_OPT</td>
<td>Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_NUM_THREADS</td>
<td>Controls the number of threads employed by the optimizer.</td>
<td>GAMS Threads</td>
</tr>
<tr>
<td>MSK_IPAR_OPTIMIZER</td>
<td>The parameter controls which optimizer is used to optimize the task.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_TIMING_LEVEL</td>
<td>Controls the amount of timing performed inside MOSEK.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_SPAR_PARAM_READ_FILE_NAME</td>
<td>Modifications to the parameter database is read from this file.</td>
<td></td>
</tr>
</tbody>
</table>

### Problem Data

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_CHECK_CONVEXITY_REL_TOL</td>
<td>This parameter controls when the full convexity check declares a problem to be non-convex.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_SYM_MAT_TOL</td>
<td>Absolute zero tolerance for elements in symmetric matrices.</td>
<td>1e-12</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_SYM_MAT_TOL_HUGE</td>
<td>An element in a symmetric matrix which is larger than this value in absolute size causes an error.</td>
<td>1e+20</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_SYM_MAT_TOL_LARGE</td>
<td>An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.</td>
<td>1e+10</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_AIJ_HUGE</td>
<td>An element in the constraint matrix which is larger than this value in absolute size causes an error.</td>
<td>1e+20</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_AIJ_LARGE</td>
<td>An element in the constraint matrix which is larger than this value in absolute size causes a warning message.</td>
<td>1e+10</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_BOUND_INF</td>
<td>Any bound which in absolute value is greater than this parameter is considered infinite.</td>
<td>1e+16</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_BOUND_WRN</td>
<td>If a bound value is larger than this value in absolute size, then a warning message is issued.</td>
<td>1e+08</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_CJ_LARGE</td>
<td>A coefficient in the objective function which is larger than this value in absolute terms causes a warning message.</td>
<td>1e+08</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_C_HUGE</td>
<td>A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error.</td>
<td>1e+16</td>
</tr>
</tbody>
</table>
## 5.27 MOSEK

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_DATA_TOL_QIJ</td>
<td>Absolute zero tolerance for coefficients of quadratic terms.</td>
<td>1e-16</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_X</td>
<td>Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_LOWER_OBJ_CUT</td>
<td>Lower objective limit.</td>
<td>-1e+30</td>
</tr>
<tr>
<td>MSK_DPAR_LOWER_OBJ_CUT_FINE_TRIH</td>
<td>Lower objective limit threshold.</td>
<td>-5e+29</td>
</tr>
<tr>
<td>MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL</td>
<td>This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.</td>
<td>1e-15</td>
</tr>
<tr>
<td>MSK_DPAR_UPPER_OBJ_CUT</td>
<td>Upper objective limit.</td>
<td>1e+30</td>
</tr>
<tr>
<td>MSK_DPAR_UPPER_OBJ_CUT_FINE_TRIH</td>
<td>Upper objective limit threshold.</td>
<td>5e+29</td>
</tr>
<tr>
<td>MSK_IPAR_CHECK_CONVEXITY</td>
<td>Specify the level of convexity check on quadratic problems.</td>
<td>MSK_CHECK_CONVEXITY_FULL</td>
</tr>
<tr>
<td>QEXTRACTALG</td>
<td>Switch to choose extraction algorithm for quadratic equations in GAMS interface.</td>
<td>0</td>
</tr>
<tr>
<td>SDPCHECKVARS</td>
<td>Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.27.2.3 Presolving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP</td>
<td>Absolute tolerance employed by the linear dependency checker.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_AIJ</td>
<td>Absolute zero tolerance employed for constraint coefficients in presolve.</td>
<td>1e-12</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_REL_LINDEP</td>
<td>Relative tolerance employed by the linear dependency checker.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_S</td>
<td>Absolute zero tolerance employed for dual variables in presolve.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_X</td>
<td>Absolute zero tolerance employed for primal variables in presolve.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL</td>
<td>Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_TRIES</td>
<td>Control the maximum number of times the eliminator is tried.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRI</td>
<td>Controls the linear dependency check, which is potentially computationally expensive.</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_REL_TRH</td>
<td>Controls the linear dependency check, which is potentially computationally expensive.</td>
<td>100</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_USE</td>
<td>Controls whether the linear constraints are checked for linear dependencies.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_MAX_NUM_PASS</td>
<td>Control the maximum number of times presolve passes over the problem.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS</td>
<td>Controls the maximum number of reductions performed by the presolve.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_USE</td>
<td>Controls whether the presolve is applied to a problem before it is optimized.</td>
<td>MSK_PRESOLVE_MODE_FREE</td>
</tr>
</tbody>
</table>

### 5.27.2.4 Simplex Optimizer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_BASIS_REL_TOL_S</td>
<td>Maximum relative dual bound violation allowed in an optimal basic solution.</td>
<td>1e-12</td>
</tr>
<tr>
<td>MSK_DPAR_BASIS_TOL_S</td>
<td>Maximum absolute dual bound violation in an optimal basic solution.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_DPAR_BASIS_TOL_X</td>
<td>Maximum absolute primal bound violation allowed in an optimal basic solution.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_DPAR_SIMPLEX_ABS_TOL_Pivot</td>
<td>Absolute pivot tolerance employed by the simplex optimizers.</td>
<td>1e-07</td>
</tr>
<tr>
<td>MSK_DPAR_SIM_LU_TOL_REL</td>
<td>Relative pivot tolerance for LU factorization in simplex optimizers and basis identification.</td>
<td>0.01</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_BASIS_FACTOR_USE</td>
<td>Controls whether an LU factorization of the basis is used in a hot-start.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_DEGEN</td>
<td>Controls how aggressively degeneration is handled.</td>
<td>MSK_SIM_DEGEN_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_DUAL_CRASH</td>
<td>Controls whether crashing is performed in the dual simplex optimizer.</td>
<td>90</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION</td>
<td>Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex.</td>
<td>50</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_DUAL_SELECTION</td>
<td>Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.</td>
<td>MSK_SIM_SELECTION_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_EXPLOIT_DUPVEC</td>
<td>Controls if the simplex optimizers are allowed to exploit duplicated columns.</td>
<td>MSK_SIM_EXPLOIT_DUPVEC_OFF</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_HOTSTART</td>
<td>Controls the type of hot-start that the simplex optimizer perform.</td>
<td>MSK_SIM_HOTSTART_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_HOTSTART_LU</td>
<td>Determines if the simplex optimizer should exploit the initial factorization.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_MAX_ITERATIONS</td>
<td>Maximum number of iterations that can be used by a simplex optimizer.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_MAX_NUM_SETBACKS</td>
<td>Controls how many set-backs are allowed within a simplex optimizer.</td>
<td>250</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_NON_SINGULAR</td>
<td>Controls if the simplex optimizer ensures a non-singular basis, if possible.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_CRASH</td>
<td>Controls whether crashing is performed in the primal simplex optimizer.</td>
<td>90</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION</td>
<td>Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex.</td>
<td>50</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_SELECTION</td>
<td>Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.</td>
<td>MSK_SIM_SELECTION_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_REFORMULATION</td>
<td>Controls if the simplex optimizers are allowed to reformulate the problem.</td>
<td>MSK_SIM_REFORMULATION_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SAVE_LU</td>
<td>Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SCALING</td>
<td>Controls how much effort is used in scaling the problem before a simplex optimizer is used.</td>
<td>MSK_SCALING_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SCALING_METHOD</td>
<td>Controls how the problem is scaled before a simplex optimizer is used.</td>
<td>MSK_SCALING_METHOD_POW2</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SEED</td>
<td>Sets the random seed used for randomization in the simplex optimizers.</td>
<td>23456</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SOLVE_Form</td>
<td>Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.</td>
<td>MSK_SOLVE_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_STABILITY_PRIORITY</td>
<td>Controls how high priority the numerical stability should be given.</td>
<td>50</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SWITCH_OPTIMIZER</td>
<td>Controls the simplex behavior.</td>
<td>MSK_OFF</td>
</tr>
</tbody>
</table>

5.27.2.5 Interior Point Optimizer and Basis Identification
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_DFEAS</td>
<td>Dual feasibility tolerance used by the interior-point optimizer for conic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_INFEAS</td>
<td>Infeasibility tolerance used by the interior-point optimizer for conic problems.</td>
<td>1e-12</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_MU_RED</td>
<td>Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_NEAR_REL</td>
<td>Termination tolerance multiplier that is used if no accurate solution can be found.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_PFEAS</td>
<td>Primal feasibility tolerance used by the interior-point optimizer for conic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_REL_GAP</td>
<td>Relative gap termination tolerance used by the interior-point optimizer for conic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_DFEAS</td>
<td>Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_INFEAS</td>
<td>Infeasibility tolerance used by the interior-point optimizer for quadratic problems.</td>
<td>1e-12</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_MU_RED</td>
<td>Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_NEAR_REL</td>
<td>Termination tolerance multiplier that is used if no accurate solution can be found.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_PFEAS</td>
<td>Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_QO_TOL_REL_GAP</td>
<td>Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_DFEAS</td>
<td>Dual feasibility tolerance used by the interior-point optimizer for linear problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_DSAFE</td>
<td>Controls the initial dual starting point used by the interior-point optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_INFEAS</td>
<td>Infeasibility tolerance used by the interior-point optimizer for linear problems.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_MU_RED</td>
<td>Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.</td>
<td>1e-16</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PATH</td>
<td>Controls how close the interior-point optimizer follows the central path.</td>
<td>1e-08</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PFEAS</td>
<td>Primal feasibility tolerance used by the interior-point optimizer for linear problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PSAFE</td>
<td>Controls the initial primal starting point used by the interior-point optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_REL_GAP</td>
<td>Relative gap termination tolerance used by the interior-point optimizer for linear problems.</td>
<td>1e-08</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_REL_STEP</td>
<td>Relative step size to the boundary for linear and quadratic optimization problems.</td>
<td>0.9999</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_STEP_SIZE</td>
<td>Step size tolerance.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_IPAR_BI_CLEAN_OPTIMIZER</td>
<td>Controls which simplex optimizer is used in the clean-up phase.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_BI_IGNORE_MAX_ITER</td>
<td>Controls if basis identification is performed under certain conditions.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_BI_IGNORE_NUM_ERROR</td>
<td>Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_BI_MAX_ITERATIONS</td>
<td>Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.</td>
<td>1000000</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_BASIS</td>
<td>Controls whether the interior-point optimizer also computes an optimal basis.</td>
<td>MSK_BI_ALWAYS</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_DIFF_STEP</td>
<td>Controls whether different step sizes are allowed in the primal and dual space.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_ITERATIONS</td>
<td>Controls the maximum number of iterations allowed in the interior-point optimizer.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_NUM_COR</td>
<td>Controls the maximum number of correctors allowed by the multiple corrector procedure.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS</td>
<td>Maximum number of steps to be used by the iterative refinement of the search direction.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MULTI_THREAD</td>
<td>Controls whether the interior-point optimizers can employ multiple threads if available.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_OFF_COL_TRH</td>
<td>Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix.</td>
<td>40</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS</td>
<td>The GP ordering is dependent on a random seed.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_ORDER_METHOD</td>
<td>Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.</td>
<td>MSK_ORDER_METHOD_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_REGULARIZATION_USE</td>
<td>Controls whether regularization is allowed.</td>
<td>MSK_ON</td>
</tr>
</tbody>
</table>
### 5.27.2.6 Mixed Integer Optimizer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXOPTFILE</td>
<td>Name of option file which is read just before solving the fixed problem.</td>
<td></td>
</tr>
<tr>
<td>MSK_DPAR_MIO_MAX_TIME</td>
<td>This parameter limits the maximum time spent by the mixed-integer optimizer.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_REL_GAP_CONST</td>
<td>This value is used to compute the relative gap for the solution to an integer optimization problem.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_ABS_GAP</td>
<td>Absolute optimality tolerance employed by the mixed-integer optimizer.</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_ABS_RELAX</td>
<td>Absolute relaxation tolerance of the integer constraints.</td>
<td>1e-05</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_FEAS</td>
<td>Feasibility tolerance for mixed integer solver.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_REL_DUAL</td>
<td>If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_REL_GAP</td>
<td>Relative optimality tolerance employed by the mixed-integer optimizer.</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_BRANCH_DIR</td>
<td>Controls whether the mixed-integer optimizer is branching up or down by default.</td>
<td>MSK_BRANCH_DIR_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CONIC_OUTER</td>
<td>If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_CLIQUE</td>
<td>Controls whether clique cuts should be generated.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_CMIR</td>
<td>Controls whether mixed integer rounding cuts should be generated.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_GMI</td>
<td>Controls whether GMI cuts should be generated.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUTIMPLIED_BOUND</td>
<td>Controls whether implied bound cuts should be generated.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_KNAPSACKCOVER</td>
<td>Controls whether knapsack cover cuts should be generated.</td>
<td>MSK_OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_IPAR_INTPNT_SCALING</td>
<td>Controls how the problem is scaled before the interior-point optimizer is used.</td>
<td>MSK_SCALING_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_SOLVE_FORM</td>
<td>Controls whether the primal or the dual problem is solved.</td>
<td>MSK_SOLVE_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_STARTING_POINT</td>
<td>Starting point used by the interior-point optimizer.</td>
<td>MSK_STARTING_POINT_FREE</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_CUT_SELECTION_LEVEL</code></td>
<td>Controls how aggressively generated cuts are selected to be included in the relaxation.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_FEASPUMP_LEVEL</code></td>
<td>Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_HEURISTIC_LEVEL</code></td>
<td>Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_MAX_NUM Branches</code></td>
<td>Maximum number of branches allowed during the branch and bound search.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_MAX_NUM_RELAXS</code></td>
<td>Maximum number of relaxations allowed during the branch and bound search.</td>
<td>GAMS NodLim</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS</code></td>
<td>Maximum number of cut separation rounds at the root node.</td>
<td>100</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_MAX_NUM_SOLUTIONS</code></td>
<td>The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_NODE_OPTIMIZER</code></td>
<td>Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_NODE_SELECTION</code></td>
<td>Controls the node selection strategy employed by the mixed-integer optimizer.</td>
<td>MSK_MIO_NODE_SELECTION_FREE</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE</code></td>
<td>Enables or disables perspective reformulation in presolve.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_PROBING_LEVEL</code></td>
<td>Controls the amount of probing employed by the mixed-integer optimizer in presolve.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT</code></td>
<td>Use objective domain propagation.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_RINS_MAX_NODES</code></td>
<td>Controls the maximum number of nodes allowed in each call to the RINS heuristic.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_ROOT_OPTIMIZER</code></td>
<td>Controls which optimizer is employed at the root node in the mixed-integer optimizer.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL</code></td>
<td>Controls whether presolve can be repeated at root node.</td>
<td>-1</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_SEED</code></td>
<td>Sets the random seed used for randomization in the mixed integer optimizer.</td>
<td>42</td>
</tr>
<tr>
<td><code>MSK_IPAR_MIO_VB_DETECTION_LEVEL</code></td>
<td>Controls how much effort is put into detecting variable bounds.</td>
<td>-1</td>
</tr>
<tr>
<td><code>SOLVEFINAL</code></td>
<td>Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished.</td>
<td>1</td>
</tr>
</tbody>
</table>

5.27.2.7 Infeasibility Analyser for Continuous Problems
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_ANA_SOL_INFEAS_TOL</td>
<td>If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.</td>
<td>1e-06</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_PREFER_PRIMAL</td>
<td>If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_REPORT_AUTO</td>
<td>Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_REPORT_LEVEL</td>
<td>Controls the amount of information presented in an infeasibility report.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.27.2.8 Output

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_IPAR_LOG</td>
<td>Controls the amount of log information.</td>
<td>10</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_BI</td>
<td>Controls the amount of output printed by the basis identification procedure.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_BI_FREQ</td>
<td>Controls logging frequency of the basis identification</td>
<td>2500</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_CHECK_CONVEXITY</td>
<td>Controls logging in convexity check on quadratic problems.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_FEAS_REPAIR</td>
<td>Controls the amount of output printed when performing feasibility repair.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_INFEAS_ANA</td>
<td>Controls amount of output printed by the infeasibility analyzer procedures.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_INTPNT</td>
<td>Controls amount of output printed by the interior-point optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_MIO</td>
<td>Controls the log level for the mixed-integer optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_MIO_FREQ</td>
<td>Controls how frequent the mixed-integer optimizer prints the log line.</td>
<td>10</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_ORDER</td>
<td>If turned on, then factor lines are added to the log.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_PRESOLVE</td>
<td>Controls amount of output printed by the presolve procedure.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_RESPONSE</td>
<td>Controls amount of output printed when response codes are reported.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_SIM</td>
<td>Controls amount of output printed by the simplex optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_SIM_FREQ</td>
<td>Controls simplex optimizer logging frequency.</td>
<td>1000</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_STORAGE</td>
<td>When turned on, MOSEK prints messages regarding the storage usage and allocation.</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_HEADER</code></td>
<td>Write a text header with date and MOSEK version in an OPF file.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_HINTS</code></td>
<td>Write a hint section with problem dimensions in the beginning of an OPF file.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_LINE_LENGTH</code></td>
<td>Aim to keep lines in OPF files not much longer than this.</td>
<td>80</td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_PARAMETERS</code></td>
<td>Write a parameter section in an OPF file.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_PROBLEM</code></td>
<td>Write objective, constraints, bounds etc.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_SOLUTIONS</code></td>
<td>Enable inclusion of solutions in the OPF files.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_SOL_BAS</code></td>
<td>Whether to include basic solution in OPF files.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_SOL_ITG</code></td>
<td>Whether to include integer solution in OPF files.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_OPF_WRITE_SOL_ITR</code></td>
<td>Whether to include interior solution in OPF files.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_PTF_WRITE_TRANSFORM</code></td>
<td>If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_COMPRESSION</code></td>
<td>Controls whether the data file is compressed while it is written.</td>
<td>9</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_DATA_PARAM</code></td>
<td>If this option is turned on the parameter settings are written to the data file as parameters.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE GENERIC_NAMES</code></td>
<td>Controls whether generic names should be used instead of user-defined names when writing to the data file.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE GENERIC_NAMES_IO</code></td>
<td>Index origin used in generic names.</td>
<td>1</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS</code></td>
<td>Controls if the writer ignores incompatible problem items when writing files.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP_FULL_OBJ</code></td>
<td>Write all variables, including the ones with 0-coefficients, in the objective.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP_LINE_WIDTH</code></td>
<td>Maximum width of line in an LP file written by MOSEK.</td>
<td>80</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP QUOTED NAMES</code></td>
<td>If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.</td>
<td><code>MSK_ON</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP STRICT FORMAT</code></td>
<td>Controls whether LP output files satisfy the LP format strictly.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP TERMS_PER_LINE</code></td>
<td>Maximum number of terms on a single line in an LP file written by MOSEK.</td>
<td>10</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_MPS_FORMAT</code></td>
<td>Controls in which format the MPS is written.</td>
<td><code>MSK_MPS_FORMAT_FREE</code></td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_IPAR_WRITE_MPS_INT</strong></td>
<td>Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.</td>
<td><strong>MSK_ON</strong></td>
</tr>
<tr>
<td><strong>MSK_IPAR_WRITE_PRECISION</strong></td>
<td>Controls the precision with which double numbers are printed in the MPS data file.</td>
<td>15</td>
</tr>
<tr>
<td><strong>MSK_IPAR_WRITE_TASK_INCLUSION</strong></td>
<td>Controls whether the solutions are stored in the task file too.</td>
<td><strong>MSK_ON</strong></td>
</tr>
<tr>
<td><strong>MSK_IPAR_WRITE_XML_MODE</strong></td>
<td>Controls if linear coefficients should be written by row or column when writing in the XML file format.</td>
<td><strong>MSK_WRITE_XML_MODE_ROW</strong></td>
</tr>
<tr>
<td><strong>MSK_SPAR_DATA_FILE_NAME</strong></td>
<td>If set, problem data is written to this file. File extension determines format.</td>
<td></td>
</tr>
<tr>
<td><strong>MSK_SPAR_PARAM_WRITE_FILE_NAME</strong></td>
<td>The parameter database is written to this file.</td>
<td></td>
</tr>
<tr>
<td><strong>MSK_SPAR_WRITE_LP_GEN_VAR_NAME</strong></td>
<td>Sometimes when an LP file is written additional variables must be inserted.</td>
<td>&quot;xmskgen&quot;</td>
</tr>
<tr>
<td><strong>SDPSOLUFILE</strong></td>
<td>Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints.</td>
<td></td>
</tr>
</tbody>
</table>

### 5.27.3 The MOSEK Log File

The MOSEK log output gives much useful information about the current solver progress and individual phases.

#### 5.27.3.1 Log Using the Interior Point Optimizer

The following is a MOSEK log output from running the **TRNSPORT** model from the GAMS Model Library:

```
Optimizer started.  
Interior-point optimizer started.  
Presolve started.  
Linear dependency checker started.  
Linear dependency checker terminated. 
Eliminator - tries :  0  time : 0.00 
Lin. dep. - tries : 1  time : 0.00 
Lin. dep. - number : 0  
Presolve terminated. Time: 0.00  
Optimizer - threads : 1  
Optimizer - solved problem : the primal  
Optimizer - Constraints : 5  
Optimizer - Cones : 0  
Optimizer - Scalar variables : 11  conic : 0  
Optimizer - Semi-definite variables: 0  scalarized : 0  
Factor - setup time : 0.00  dense det. time : 0.00  
Factor - ML order time : 0.00  GP order time : 0.00  
Factor - nonzeros before factor : 11  after factor : 12  
Factor - dense dim. : 0  flops : 1.80e+02  
```
The first part gives information about the presolve (if used). The main log follows:

```
ITE PFEAS DFEAS GFEAS PRSTATUS POBJ DOBJ MU TIME
0 6.0e+02 1.0e+02 1.0e+02 1.00e+02 1.053000000e+00 0.000000000e+00 1.0e+02 0.00
1 5.9e+02 1.3e+02 2.6e+02 0.00e+00 3.063797526e+00 2.650041612e+02 2.4e+02 0.00
2 4.6e+01 1.0e+01 2.0e+01 -9.59e-01 3.650704301e+01 2.594819640e+02 1.9e+01 0.00
3 3.9e-01 8.7e-02 1.7e-01 -4.39e-01 1.604589379e+01 2.675438787e+02 1.6e-01 0.00
4 2.7e-02 6.0e-03 1.7e-02 9.62e-01 1.627664502e+01 1.676438787e+02 1.1e-02 0.00
5 2.2e-03 4.9e-04 9.6e-04 1.04e+00 1.585004810e+01 1.591499235e+02 8.9e-04 0.00
6 3.1e-04 6.9e-05 1.4e-04 1.01e+00 1.546312243e+01 1.547279245e+02 1.2e-04 0.00
7 2.9e-05 6.5e-06 1.3e-05 1.01e+00 1.536906429e+01 1.536999628e+02 1.2e-05 0.00
8 7.6e-08 1.7e-08 3.4e-08 1.00e+00 1.536751995e+01 1.536752387e+02 3.1e-08 0.00
9 7.5e-12 1.7e-12 3.4e-12 1.00e+00 1.536750000e+01 1.536750000e+02 3.1e-12 0.00
```

Basis identification started.

Primal basis identification phase started.
Iter. TIME
1 0.00
Primal basis identification phase terminated. Time: 0.00

Dual basis identification phase started.
Iter. TIME
0 0.00
Dual basis identification phase terminated. Time: 0.00

Basis identification terminated. Time: 0.00

Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary
Problem status: PRIMAL_AND_DUAL_FEASIBLE
Solution status: OPTIMAL
Primal. obj: 1.5367500000e+02 nrm: 6e+02 Viol. con: 3e-10 var: 0e+00
Dual. obj: 1.5367500000e+02 nrm: 2e-01 Viol. con: 0e+00 var: 5e-11

Basic solution summary
Problem status: PRIMAL_AND_DUAL_FEASIBLE
Solution status: OPTIMAL
Primal. obj: 1.5367500000e+02 nrm: 6e+02 Viol. con: 0e+00 var: 0e+00
Dual. obj: 1.5367500000e+02 nrm: 2e-01 Viol. con: 0e+00 var: 5e-11

Return code - 0 [MSK_RES_OK]: No error occurred.

The last section gives details about the model and solver status, primal and dual feasibilities, as well as solver resource times. Furthermore, the log gives information about the basis identification phase. Some of this information is listed in the GAMS solve summary in the model listing (.LST) file as well.

The fields in the main MOSEK log output are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITE</td>
<td>The number of the current iteration.</td>
</tr>
<tr>
<td>PFEAS</td>
<td>Primal feasibility.</td>
</tr>
<tr>
<td>DFEAS</td>
<td>Dual feasibility.</td>
</tr>
<tr>
<td>GFEAS</td>
<td>The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.</td>
</tr>
<tr>
<td>PRSTATUS</td>
<td>This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.</td>
</tr>
<tr>
<td>POBJ</td>
<td>Current objective function value of primal problem.</td>
</tr>
</tbody>
</table>
5.27.3.2 Log Using the Simplex Optimizer

Below is a log output running the TRNSPORT model from the GAMS Model Library using the MOSEK simplex optimizer.

Reading parameter(s) from "mosek.opt"

$ MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_DUAL_SIMPLEX

Finished reading from "mosek.opt"

Optimizer started.
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Dual simplex optimizer started.
Dual simplex optimizer setup started.
Dual simplex optimizer setup terminated.
Optimizer - solved problem : the primal
Optimizer - Constraints : 5
Optimizer - Scalar variables : 6 conic : 0
Optimizer - hotstart : no

tabular

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITER</td>
<td>Current number of iterations.</td>
</tr>
<tr>
<td>DEGITER(%)</td>
<td>Current percentage of degenerate iterations.</td>
</tr>
<tr>
<td>P/DFEAS</td>
<td>Current primal and dual infeasibility.</td>
</tr>
<tr>
<td>P/DOBJ</td>
<td>Current primal and dual objective value.</td>
</tr>
<tr>
<td>TIME</td>
<td>Current elapsed solving time in seconds.</td>
</tr>
<tr>
<td>TOTTIME</td>
<td>Total elapsed solving time in seconds.</td>
</tr>
</tbody>
</table>

Optimizer terminated. Time: 0.00

Basic solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal. obj: 1.5367500000e+02 nrm: 6e+02 Viol. con: 0e+00 var: 0e+00
Dual. obj: 1.5367500000e+02 nrm: 2e-01 Viol. con: 0e+00 var: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.
5.27.3.3 Log Using the Mixed Integer Optimizer

Below is a log output running the model CUBE from the GAMS model library using the MOSEK mixed-integer optimizer.

Optimizer started.  
Mixed integer optimizer started.  
Threads used: 1  
Presolve started.  
Presolve terminated. Time = 0.00  
Presolved problem: 76 variables, 99 constraints, 419 non-zeros  
Presolved problem: 0 general integer, 27 binary, 49 continuous  
Clique table size: 0  

<table>
<thead>
<tr>
<th>BRANCHES</th>
<th>RELAXS</th>
<th>ACT_NDS</th>
<th>DEPTH</th>
<th>BEST_INT_OBJ</th>
<th>BEST_RELAX_OBJ</th>
<th>REL_GAP(%)</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>0.0000000000e+00</td>
<td>NA</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td>100.00</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>6.0000000000e+00</td>
<td>0.0000000000e+00</td>
<td>100.00</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Cut generation started.  
Cut generation terminated. Time = 0.00  

A near optimal solution satisfying the absolute gap tolerance of 0.00e+00 has been located.  
Objective of best integer solution : 4.0000000000e+00  
Best objective bound : 3.3333333333e-01  
Construct solution objective : Not employed  
Construct solution # roundings : 0  
User objective cut value : 0  
Number of cuts generated : 3  
Number of Gomory cuts : 3  
Number of branches : 347  
Number of relaxations solved : 345  
Number of interior point iterations: 6  
Number of simplex iterations : 3460  
Time spend presolving the root : 0.00  
Time spend in the heuristic : 0.00  
Time spend in the sub optimizers : 0.00  
Time spend optimizing the root : 0.00  
Mixed integer optimizer terminated. Time: 0.12  

Optimizer terminated. Time: 0.12  

Integer solution solution summary  
Problem status : PRIMAL_FEASIBLE  
Solution status : NEAR_INTEGER_OPTIMAL  
Primal. obj: 4.0000000000e+00  nrm: 1e+01  Viol. con: 1e+01  var: 1e+00  itg: 3e-16
Return code - 10004 [MSK_RES_TRM_MIO_NEAR_ABS_GAP]: The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

The fields in the main MOSEK log output are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCHES</td>
<td>Current number of branches in tree.</td>
</tr>
<tr>
<td>RELAXS</td>
<td>Current number of nodes in branch and bound tree.</td>
</tr>
<tr>
<td>ACT_NDS</td>
<td>Current number of active nodes.</td>
</tr>
<tr>
<td>BEST_INT_OBJ</td>
<td>Current best integer solution (primal bound).</td>
</tr>
<tr>
<td>BEST_RELAX_OBJ</td>
<td>Current best relaxed solution (dual bound).</td>
</tr>
<tr>
<td>REL_GAP(%)</td>
<td>Relative gap between current BEST_INT_OBJ and BEST_RELAX_OBJ.</td>
</tr>
<tr>
<td>TIME</td>
<td>Current elapsed solving time in seconds.</td>
</tr>
</tbody>
</table>

The log then gives information about solving the model with discrete variables fixed in order to determine marginals. Option SOLVEFINAL can be used to disable this step. The fixed problem is solved as a regular LP with warm start information. So the log looks identical to the MOSEK simplex optimizer for linear programs:

Solving fixed problem...

Optimizer started.
Simplex optimizer started.
Presolve started.
Eliminator - tries : 0 time : 0.00
Lin. dep. - tries : 0 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Simplex optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Basic solution summary
   Problem status : PRIMAL_AND_DUAL_FEASIBLE
   Solution status : OPTIMAL
   Primal. obj: 4.0000000000e+00 nrm: 1e+01 Viol. con: 4e-16 var: 0e+00
   Dual. obj: 4.0000000000e+00 nrm: 5e+00 Viol. con: 0e+00 var: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

MIP Solution: 4.000000 (3466 iterations, 347 nodes)
Final Solve: 4.000000 (0 iterations)

Best possible: 0.333333
Absolute gap: 3.666667
Relative gap: 0.916667

5.27.4  Semidefinite Programming with GAMS/MOSEK (experimental)

With Version 7, MOSEK introduced a semidefinite programming (SDP) solver into their portfolio. The following introduction is taken from the Mosel Documentation.
Semidefinite programming is a generalization of quadratic conic programming, allowing the use of matrix variables belonging to the convex cone of positive semidefinite matrices

\[ S^+_r = \{ X \in S_r : z^T X z \geq 0, \ \forall z \in \mathbb{R}^r \}, \]

where \( S_r \) is the set of \( r \times r \) real-valued symmetric matrices. MOSEK can solve semidefinite optimization problems of the form

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle A_{ij}, X_j \rangle + c^f \\
\text{subject to} & \quad l^c_i \leq \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle A_{ij}, X_j \rangle \leq u^c_i, \quad i = 0, \ldots, m-1, \\
& \quad l_j^T \leq x_j \leq u_j^T, \quad j = 0, \ldots, p-1,
\end{align*}
\]

(SDP)

where the problem has \( p \) symmetric positive semidefinite (PSD) variables \( X_j \in S^+_r \) of dimension \( r_j \) with symmetric coefficient matrices \( \overline{C}_j \in S_r \), and \( \overline{A}_{ij} \in S_r \). We use the standard notation for the matrix inner product, i.e., for \( A, B \in \mathbb{R}^{m \times n} \) we have

\[ \langle A, B \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} B_{ij}. \]

### 5.27.4.1 Example

An example for a mixed semidefinite and conic quadratic programming problem with a 3-dimensional PSD matrix variable is the following:

\[
\begin{align*}
\text{minimize} & \quad \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X \rangle + x_0 \\
\text{subject to} & \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X \rangle + x_0 = 1 \\
& \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X \rangle + x_1 + x_2 = \frac{1}{2} \\
& \quad x_0 \geq \sqrt{x_1^2 + x_2^2} \\
& \quad X \succeq 0
\end{align*}
\]

The GAMS/MOSEK interface offers an experimental interface to MOSEK’s SDP solver. It allows to state SDP’s of the form (SDP) in GAMS language. For instance, the example problem from above can be formulated as follows (sdp01 in the GAMS Test Library):

```plaintext
Set i / 0 * 2 /;
alias(i, ip);
Variables barX(i,i) PSDMATRIX
x(i) simple vars
z objective var;
x.lo('0') = 0;
Parameters barAobj(i,i) coefficients of barX in objective
barAe1(i,i) coefficients of barX in e1
barAe2(i,i) coefficients of barX in e2
;
Table barAobj(i,i)
```

The GAMS/MOSEK interface offers an experimental interface to MOSEK’s SDP solver. It allows to state SDP’s of the form (SDP) in GAMS language. For instance, the example problem from above can be formulated as follows (sdp01 in the GAMS Test Library):

```plaintext
Set i / 0 * 2 /;
alias(i, ip);
Variables barX(i,i) PSDMATRIX
x(i) simple vars
z objective var;
x.lo('0') = 0;
Parameters barAobj(i,i) coefficients of barX in objective
barAe1(i,i) coefficients of barX in e1
barAe2(i,i) coefficients of barX in e2
;
Table barAobj(i,i)
```
We see that the matrix $\bar{X}$ is defined via the 2-dimensional GAMS variable $\text{barX}$. Additionally, the keyword `PSDMATRIX` at the beginning of the descriptive text (!) of the variable is used to indicate that the variables belonging to symbol $\text{barX}$ are to be interpreted as a matrix variable with PSD constraint.

The Model shown above can be solved with MOSEK via the statements

```
option qcp = mosek;
Solve m minimizing z using QCP;
```

As GAMS has no native support for conic programming, the modeltype is specified as QCP.

The solve statement produces the following log output (see also Log Using the Interior Point Optimizer):

```
M O S E K version 7.0.0.75 (Build date: 2013-7-1 19:28:43)
Copyright (C) MOSEK ApS, Fruebjergvej 3, Box 16
DK-2100 Copenhagen, Denmark
http://www.mosek.com

Recognizing SDP variables for symbols barX (1)
Recognized 1 quadratic equation as conic constraint.
Optimizer started.
Conic interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 0
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Optimizer - threads : 1
Optimizer - solved problem : the primal
Optimizer - Constraints : 2
Optimizer - Cones : 1
Optimizer - Scalar variables : 3 conic : 3
Optimizer - Semi-definite variables: 1 scalarized : 6
Factor - setup time : 0.00 dense det. time : 0.00
Factor - ML order time : 0.00 GP order time : 0.00
Factor - nonzeros before factor : 3 after factor : 3
Factor - dense dim. : 0 flops : 2.88e+02
ITE PFEAS DFEAS GFEAS PRSTATUS POBJ DOBJ MU TIME
 0  3.0e+00 1.0e+00 8.0e+00 0.00e+00 7.000000000e+00 0.000000000e+00 1.0e+00 0.00
 1  4.3e-01 1.4e-01 1.2e+00 1.67e-01 1.601234178e+00 3.103213067e-01 1.4e-01 0.00
 2  6.6e-02 2.2e-02 1.1e+00 1.21e+00 8.534948745e-01 6.726200786e-01 2.2e-02 0.00
 3  4.9e-03 1.6e-03 1.3e-02 1.02e+00 7.158760069e-01 7.026959645e-01 1.6e-03 0.00
 4  2.8e-04 9.4e-05 7.5e-04 1.00e+00 7.063055286e-01 7.055481573e-01 9.4e-05 0.00
 5  1.6e-05 5.2e-06 4.1e-05 1.00e+00 7.057440915e-01 7.057021878e-01 5.2e-06 0.00
 6  8.9e-07 3.0e-07 2.4e-06 1.00e+00 7.057124546e-01 7.057100494e-01 3.0e-07 0.00
```
5.27 MOSEK

7 1.6e-08 5.4e-09 4.3e-08 1.00e+00 7.057105296e-01 7.057104862e-01 5.4e-09 0.00
Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary
  Problem status : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal. obj: 7.0571052965e-01 Viol. con: 2e-08 var: 0e+00 barvar: 0e+00 cones: 0e+00
  Dual. obj: 7.0571048621e-01 Viol. con: 0e+00 var: 1e-16 barvar: 0e+00 cones: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

Finally, the optimal value for the matrix $\bar{X}$ and the dual values associated with bound constraints on entries of the matrix $\bar{X}$ can be displayed in GAMS as usual: display X.l, X.m;

---- 53 VARIABLE barX.L PSDMATRIX
0 1 2
0 0 0.217 0
1 -0.260 0.311 -0.260
2 0.217 -0.260 0

---- 53 VARIABLE barX.M PSDMATRIX
0 1 2
0 EPS EPS EPS
1 EPS EPS EPS
2 EPS EPS EPS

5.27.4.2 Usage

The general syntax for defining a symmetric matrix or an (indexed) set of symmetric matrices with PSD constraints in GAMS is

Variable X(a,b,c,...,i1,...,ik,i1,...,ik) "PSDMATRIX_k <explanatory text>";

The number $k > 0$ specifies the number of indices that define the row/column dimension of the matrix. Specifying only the term PSDMATRIX is equivalent to PSDMATRIX_1. For a given $k$, the $2k$-last indices are used to index the rows and columns of the matrix. The dimensions $a, b, c, ...$ are optional and can be used to index a set of matrices. In the GAMS/MOSEK output, a number in parenthesis is used to indicate the number of PSD matrices that have been found for one symbol. For example, the code

Set i / a,b /;
Set j / s01 * s42 /;
Variables z, X(j,i,i) PSDMATRIX;
generates the output

Recognizing SDP variables for symbols X (42)

if all matrices $X(j,.,.)$ also occur in a model instantiation.

Note

The current syntax for declaring PSD matrix variables via the explanatory text a variable is a temporary solution that will hopefully be replaced by a GAMS language feature in the future.

Variables that were tagged as belonging to a PSD matrix can only occur in linear constraints. Within these constraints, the coefficient matrix for a PSD matrix needs to be specified in symmetric form. That is, if the parameter matrix barAobj in the example above is equivalently specified as

Table barAobj(i,i)
0 1 2
0 2.0 0.0 0.0
1 2.0 2.0 0.0
2 0.0 2.0 2.0

the GAMS/MOSEK interface will quit with the error message
SDP coefficient matrix not symmetric: defObj_z: 2*barX(1,0) != 0*barX(0,1)

Bounds on entries in a PSD matrix variable can be specified as usual with .lo and .up attributes. These bound constraints are translated into linear constraints by the interface. If different bounds are given to symmetric entries of a PSD matrix variable (X_{i,j} vs. X_{j,i}), the stronger bounds are used, which is equivalent to adding constraints for each of the matrix entries. For nonpositive lower bounds on diagonal entries, no extra constraints are added, as they are implied by the PSD constraint.

5.27.4.3 GAMS variables vs. PSD matrix entries

As GAMS is not aware that a indexed variable will be interpreted as PSD matrix variable, it may not generate variables for all matrix entries when instantiating a model. This is especially critical if the modeler specified bounds on matrix entries that do not appear in any of the model equations, as these bounds would not be visible to the GAMS/MOSEK interface.

As an example, consider the SDP relaxation max\{-⟨W,\tilde{X}⟩\} : X_{i,i} = 1 \forall i, X \succeq 0\} corresponding to the Goemans-Williamson Randomized Approximation Algorithm for MaxCut:

Parameter X(i,i) edge weights;  
Variable X(i,j) PSDMATRIX  
objective var;  
Equation sdpobj;  
sdpobj.. sum((i,j), -W(i,j)*(X(i,j) + X(j,i))) =e= sdpobjvar;  
X.fx(i,i) = 1.0;  
model maxcutsdp / all /;  

As W(i,i)=0, GAMS will not create any variables for X(i,i) when instantiating the model in a solve statement. As a consequence, the constraints X(i,i)=1 will not be visible to GAMS/MOSEK. However, as MOSEK will compute values for the full matrix X, it will also compute values for the diagonal entries of X.

Further, the GAMS/MOSEK interface can pass primal solution values only for those entries of a PSD matrix variable that have a corresponding GAMS variable. This may make a solution matrix appear not to be PSD in GAMS, because not all entries have been passed back.

To be aware of such problems, the GAMS/MOSEK interface checks that it has a GAMS variable available for every entry of a PSD matrix variable. If not, it will report an error like

ERROR: Have 1600 GAMS variables for entries of 400 x 400 PSD Variable X(,), expected 160000 many.

This check can be disabled by setting the option SDPCHECKVARS to 0. Note, that the check is not able to alarm the user in situations where no GAMS variables were created for all entries of a row and corresponding column.

A simple workaround for this issue is to force all variables to be generated when a model is instantiated. This can be done by adding something like eps*sum((i,j),X(i,j)) to one of the equations. Note, that eps is numerically equal to a 0.0 in GAMS, but has the effect that the term sum((i,j),X(i,j)) is passed to the solver with 0.0-coefficient.

5.27.4.4 Dual Values for PSD constraints

The PSD constraint on a matrix variable \tilde{X} is associated with a dual PSD matrix variable \tilde{Y}. As GAMS is not aware of the PSD constraints, it is also not aware of the corresponding dual variables. Thus, there is no native way to pass the duals for the PSD constraints back to GAMS. (Recall, that the marginals for a PSDMATRIX variable X are used to store the dual values associated with the bound constraints on matrix entries.)

To work around this issue, the GAMS/MOSEK interface offers the option SDPSOLUFILE. This option allows to specify the name of a GDX file that stores primal values for all entries of a matrix variable and dual values of the corresponding PSD constraint. For a variable X(i,j), the GDX file stores the primal matrix value for all entries (i,j) (i.e., not just the ones for which GAMS variables were created, therefor offering another workaround for the issue discussed in the previous section) as level values of variable X and the dual matrix for the PSD constraint as marginal values.
5.27.4.5 Infeasible and Unbounded SDPs

The GAMS/Mosek link currently does not pass on certificates for primal or dual infeasibility from Mosek to GAMS if PSD variables are present.

5.27.5 Detailed Descriptions of MOSEK Options

**FIXOPTFILE (string):** Name of option file which is read just before solving the fixed problem.

**MSK_DPAR_ANA_SOL_INFEAS_TOL (real):** If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default: 1e-06

**MSK_DPAR_BASIS_REL_TOL_S (real):** Maximum relative dual bound violation allowed in an optimal basic solution.

Default: 1e-12

**MSK_DPAR_BASIS_TOL_S (real):** Maximum absolute dual bound violation in an optimal basic solution.

Range: [1e-09, ∞]

Default: 1e-06

**MSK_DPAR_BASIS_TOL_X (real):** Maximum absolute primal bound violation allowed in an optimal basic solution.

Range: [1e-09, ∞]

Default: 1e-06

**MSK_DPAR_CHECK_CONVEXITY_REL_TOL (real):** This parameter controls when the full convexity check declares a problem to be non-convex.

Increasing this tolerance relaxes the criteria for declaring the problem non-convex. A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated. The matrix $Q$ is considered to not be PSD if the pivot element for column $i$ is less or equal to $-\|Q_{ii}\| \cdot MSK_DPAR_CHECK_CONVEXITY_REL_TOL$.

Default: 1e-10

**MSK_DPAR_DATA_SYM_MAT_TOL (real):** Absolute zero tolerance for elements in symmetric matrices.

If any value in a symmetric matrix is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Range: [1e-16, 1e-06]

Default: 1e-12
MSK_DPAR_DATA_SYM_MAT_TOL_HUGE (real): An element in a symmetric matrix which is larger than this value in absolute size causes an error. ↔

Default: 1e+20

MSK_DPAR_DATA_SYM_MAT_TOL_LARGE (real): An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed. ↔

Default: 1e+10

MSK_DPAR_DATA_TOL_AIJ_HUGE (real): An element in the constraint matrix which is larger than this value in absolute size causes an error. ↔

Default: 1e+20

MSK_DPAR_DATA_TOL_AIJ_LARGE (real): An element in the constraint matrix which is larger than this value in absolute size causes a warning message. ↔

Default: 1e+10

MSK_DPAR_DATA_TOL_BOUND_INF (real): Any bound which in absolute value is greater than this parameter is considered infinite. ↔

Default: 1e+16

MSK_DPAR_DATA_TOL_BOUND_WRN (real): If a bound value is larger than this value in absolute size, then a warning message is issued. ↔

Default: 1e+08

MSK_DPAR_DATA_TOL_CJ_LARGE (real): A coefficient in the objective function which is larger than this value in absolute terms causes a warning message. ↔

Default: 1e+08

MSK_DPAR_DATA_TOL_C_HUGE (real): A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error. ↔

Default: 1e+16

MSK_DPAR_DATA_TOL_QIJ (real): Absolute zero tolerance for coefficients of quadratic terms. ↔

Default: 1e-16

MSK_DPAR_INTPNT_CO_TOL_DFEAS (real): Dual feasibility tolerance used by the interior-point optimizer for conic problems. ↔

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.
**MSK_DPAR_INTPNT_CO_TOL_INFEAS** *(real):* Infeasibility tolerance used by the interior-point optimizer for conic problems.  

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: [0, 1]

Default: 1e-12

**MSK_DPAR_INTPNT_CO_TOL_MU_RED** *(real):* Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.  

Range: [0, 1]

Default: 1e-08

**MSK_DPAR_INTPNT_CO_TOL_NEAR_REL** *(real):* Termination tolerance multiplier that is used if no accurate solution can be found.  

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: [1, \(\infty\)]

Default: 1

**MSK_DPAR_INTPNT_CO_TOL_PFEAS** *(real):* Primal feasibility tolerance used by the interior-point optimizer for conic problems.  

Range: [0, 1]

Default: 1e-08

See also: **MSK_DPAR_INTPNT_CO_TOL_NEAR_REL**.

**MSK_DPAR_INTPNT_CO_TOL_REL_GAP** *(real):* Relative gap termination tolerance used by the interior-point optimizer for conic problems.  

Range: [0, 1]

Default: 1e-08

See also: **MSK_DPAR_INTPNT_CO_TOL_NEAR_REL**.

**MSK_DPAR_INTPNT_QO_TOL_DFEAS** *(real):* Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.  

Range: [0, 1]

Default: 1e-08

See also: **MSK_DPAR_INTPNT_QO_TOL_NEAR_REL**.

**MSK_DPAR_INTPNT_QO_TOL_INFEAS** *(real):* Infeasibility tolerance used by the interior-point optimizer for quadratic problems.  


Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: \([0, 1]\)
Default: \(1e^{-12}\)

**MSK_DPAR_INTPNT_QO_TOL_MU_RED (real)**: Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.

Range: \([0, 1]\)
Default: \(1e^{-08}\)

**MSK_DPAR_INTPNT_QO_TOL_NEAR_REL (real)**: Termination tolerance multiplier that is used if no accurate solution can be found.

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: \([1, \infty]\)
Default: 1

**MSK_DPAR_INTPNT_QO_TOL_PFEAS (real)**: Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.

Range: \([0, 1]\)
Default: \(1e^{-08}\)

See also: **MSK_DPAR_INTPNT_QO_TOL_NEAR_REL**.

**MSK_DPAR_INTPNT_QO_TOL_REL_GAP (real)**: Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.

Range: \([0, 1]\)
Default: \(1e^{-08}\)

See also: **MSK_DPAR_INTPNT_QO_TOL_NEAR_REL**.

**MSK_DPAR_INTPNT_TOL_DFEAS (real)**: Dual feasibility tolerance used by the interior-point optimizer for linear problems.

Range: \([0, 1]\)
Default: \(1e^{-08}\)

**MSK_DPAR_INTPNT_TOL_DSAFE (real)**: Controls the initial dual starting point used by the interior-point optimizer.

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: \([0.0001, \infty]\)
Default: 1
**MSK_DPAR_INTPNT_TOL_INFEAS** (*real*): Infeasibility tolerance used by the interior-point optimizer for linear problems.  

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: \([0, 1]\)

Default: \(1e^{-10}\)

**MSK_DPAR_INTPNT_TOL_MU_RED** (*real*): Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Range: \([0, 1]\)

Default: \(1e^{-16}\)

**MSK_DPAR_INTPNT_TOL_PATH** (*real*): Controls how close the interior-point optimizer follows the central path.

A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Range: \([0, 0.999]\)

Default: \(1e^{-08}\)

**MSK_DPAR_INTPNT_TOL_PFEAS** (*real*): Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Range: \([0, 1]\)

Default: \(1e^{-08}\)

**MSK_DPAR_INTPNT_TOL_PSAFE** (*real*): Controls the initial primal starting point used by the interior-point optimizer.

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Range: \([0.0001, \infty]\)

Default: 1

**MSK_DPAR_INTPNT_TOL_REL_GAP** (*real*): Relative gap termination tolerance used by the interior-point optimizer for linear problems.

Range: \([1e^{-14}, \infty]\)

Default: \(1e^{-08}\)

**MSK_DPAR_INTPNT_TOL_REL_STEP** (*real*): Relative step size to the boundary for linear and quadratic optimization problems.

Range: \([0.0001, 0.999999]\)

Default: 0.9999

**MSK_DPAR_INTPNT_TOL_STEP_SIZE** (*real*): Step size tolerance.
If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Range: [0, 1]
Default: 1e-06

**MSK_DPAR_LOWER_OBJ_CUT** *(real)*: Lower objective limit. →

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval \((\text{MSK\_DPAR\_LOWER\_OBJ\_CUT}, \text{MSK\_DPAR\_UPPER\_OBJ\_CUT})\), then MOSEK is terminated.

Range: \([-\infty, \infty]\]
Default: -1e+30

See also: **MSK_DPAR_LOWER_OBJ_CUT\_FINITE\_TRH**.

**MSK_DPAR_LOWER_OBJ_CUT\_FINITE\_TRH** *(real)*: Lower objective limit threshold. →

If the lower objective cut (\(\text{MSK\_DPAR\_LOWER\_OBJ\_CUT}\)) is less than this value, then it is treated as infinity.

Range: \([-\infty, \infty]\]
Default: -5e+29

**MSK_DPAR_MIO_MAX\_TIME** *(real)*: This parameter limits the maximum time spent by the mixed-integer optimizer. →

A negative number means infinity.

Range: \([-\infty, \infty]\]
Default: -1

**MSK_DPAR_MIO_REL\_GAP\_CONST** *(real)*: This value is used to compute the relative gap for the solution to an integer optimization problem. →

Range: \([1e-15, \infty]\]
Default: 1e-10

**MSK_DPAR_MIO_TOL\_ABS\_GAP** *(real)*: Absolute optimality tolerance employed by the mixed-integer optimizer. →

Default: GAMS OptCA

**MSK_DPAR_MIO_TOL\_ABS\_RELAX\_INT** *(real)*: Absolute relaxation tolerance of the integer constraints. →

That means if the fractional part of a discrete variable \(\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)\) is less than the tolerance, then the integer restriction is assumed to be satisfied.

Range: \([1e-09, \infty]\]
Default: 1e-05
MSK_DPAR_MIO_TOL_FEAS (real): Feasibility tolerance for mixed integer solver. 

Range: [1e-09, 0.001]
Default: 1e-06

MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT (real): If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.

A value of 0.0 means that the value is selected automatically.

Range: [0, 1]
Default: 0

MSK_DPAR_MIO_TOL_REL_GAP (real): Relative optimality tolerance employed by the mixed-integer optimizer.

Default: GAMS OptCR

MSK_DPAR_OPTIMIZER_MAX_TIME (real): Maximum amount of time the optimizer is allowed to spent on the optimization.

A negative number means infinity.

Range: [-∞, ∞]
Default: GAMS ResLim

MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP (real): Absolute tolerance employed by the linear dependency checker.

Default: 1e-06

MSK_DPAR_PRESOLVE_TOL_AIJ (real): Absolute zero tolerance employed for constraint coefficients in presolve.

Range: [1e-15, ∞]
Default: 1e-12

MSK_DPAR_PRESOLVE_TOL_REL_LINDEP (real): Relative tolerance employed by the linear dependency checker.

Default: 1e-10

MSK_DPAR_PRESOLVE_TOL_S (real): Absolute zero tolerance employed for dual variables in presolve.

Default: 1e-08

MSK_DPAR_PRESOLVE_TOL_X (real): Absolute zero tolerance employed for primal variables in presolve.

Default: 1e-08

MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL (real): This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.
Default: $1e^{-15}$

**MSK_DPAR_SEMIDEFINITE_TOL_APPROX** (real): Tolerance to define a matrix to be positive semidefinite. ←

Range: $[1e^{-15}, \infty]$  
Default: $1e^{-10}$

**MSK_DPAR_SIMPLEX_ABS_TOL_PIV** (real): Absolute pivot tolerance employed by the simplex optimizers. ←

Range: $[1e^{-12}, \infty]$  
Default: $1e^{-07}$

**MSK_DPAR_SIM_LU_TOL_REL_PIV** (real): Relative pivot tolerance for LU factorization in simplex optimizers and basis identification. ←

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Range: $[1e^{-06}, 0.999999]$  
Default: 0.01

**MSK_DPAR_UPPER_OBJ_CUT** (real): Upper objective limit. ←

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval ($MSK_DPAR_LOWER_OBJ_CUT$, $MSK_DPAR_UPPER_OBJ_CUT$), then MOSEK is terminated.

Range: $[-\infty, \infty]$  
Default: $1e+30$

See also: **MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH**.

**MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH** (real): Upper objective limit threshold. ←

If the upper objective cut $MSK_DPAR_UPPER_OBJ_CUT$ is greater than this value, then it is treated as infinity.

Range: $[-\infty, \infty]$  
Default: $5e+29$

**MSK_IPAR_AUTO_SORT_A_BEFORE_OPT** (string): Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed. ←

This is not required but makes the optimization insusceptible to reorderings of variables.

Default: **MSK_OFF**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_ON</strong></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><strong>MSK_OFF</strong></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_BI_CLEAN_OPTIMIZER** *(string)*: Controls which simplex optimizer is used in the clean-up phase.

Anything else than primal or dual simplex is equivalent to free simplex.

Default: MSK_OPTIMIZER_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_OPTIMIZER_FREE</td>
<td>The optimizer is chosen automatically.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_PRIMAL_SIMPLEX</td>
<td>The primal simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_DUAL_SIMPLEX</td>
<td>The dual simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_FREE_SIMPLEX</td>
<td>One of the simplex optimizers is used.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_BI_IGNORE_MAX_ITER** *(string)*: Controls if basis identification is performed under certain conditions.

If the parameter MSK_IPAR_INTPNT_BASIS has the value MSK_BI_NO_ERROR and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value MSK_ON.

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_BI_IGNORE_NUM_ERROR** *(string)*: Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.

If the parameter MSK_IPAR_INTPNT_BASIS has the value MSK_BI_NO_ERROR and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value MSK_ON.

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_BI_MAX_ITERATIONS** *(integer)*: Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default: 1000000

**MSK_IPAR_CHECK_CONVEXITY** *(string)*: Specify the level of convexity check on quadratic problems.

Default: MSK_CHECK_CONVEXITY_FULL

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_CHECK_CONVEXITY_NONE</td>
<td>No convexity check.</td>
</tr>
<tr>
<td>MSK_CHECK_CONVEXITY_SIMPLE</td>
<td>Perform simple and fast convexity check.</td>
</tr>
<tr>
<td>MSK_CHECK_CONVEXITY_FULL</td>
<td>Perform a full convexity check.</td>
</tr>
</tbody>
</table>
MSK_IPAR_INFEAS_PREFER_PRIMAL (string): If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on. ↔

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

MSK_IPAR_INFEAS_REPORT_AUTO (string): Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible. ↔

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

MSK_IPAR_INFEAS_REPORT_LEVEL (integer): Controls the amount of information presented in an infeasibility report. ↔

Higher values imply more information.

Default: 1

MSK_IPAR_INTPNT_BASIS (string): Controls whether the interior-point optimizer also computes an optimal basis. ↔

Default: MSK_BI_ALWAYS

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_BI_NEVER</td>
<td>Never do basis identification.</td>
</tr>
<tr>
<td>MSK_BI_ALWAYS</td>
<td>Basis identification is always performed even if the interior-point optimizer terminates abnormally.</td>
</tr>
<tr>
<td>MSK_BI_NO_ERROR</td>
<td>Basis identification is performed if the interior-point optimizer terminates without an error.</td>
</tr>
<tr>
<td>MSK_BI_IF_FEASIBLE</td>
<td>Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.</td>
</tr>
</tbody>
</table>

See also: MSK_IPAR_BI_CLEAN_OPTIMIZER, MSK_IPAR_BI_IGNORE_MAX_ITER, MSK_IPAR_BI_IGNORE_NUM_ERROR, MSK_IPAR_BI_MAX_ITERATIONS.

MSK_IPAR_INTPNT_DIFF_STEP (string): Controls whether different step sizes are allowed in the primal and dual space. ↔

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_INTPNT_MAX_ITERATIONS** (integer): Controls the maximum number of iterations allowed in the interior-point optimizer. ←

Default: GAMS IterLim

**MSK_IPAR_INTPNT_MAX_NUM_COR** (integer): Controls the maximum number of correctors allowed by the multiple corrector procedure. ←

A negative value means that MOSEK is making the choice.

Range: \{-1, ..., \infty\}

Default: -1

**MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS** (integer): Maximum number of steps to be used by the iterative refinement of the search direction. ←

A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

Range: \{-\infty, ..., \infty\}

Default: -1

**MSK_IPAR_INTPNT_MULTI_THREAD** (string): Controls whether the interior-point optimizers can employ multiple threads if available. ←

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_INTPNT_OFF_COL_TRH** (integer): Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix. ←

0 means no detection, 1 means aggressive detection, and higher values mean less aggressive detection.

Default: 40

**MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS** (integer): The GP ordering is dependent on a random seed. ←

Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried. A value of 0 means that MOSEK makes the choice.

Default: 0

**MSK_IPAR_INTPNT_ORDER_METHOD** (string): Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system. ←

Default: MSK_ORDER_METHOD_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ORDER_METHOD_FREE</td>
<td>The ordering method is chosen automatically.</td>
</tr>
</tbody>
</table>
**value** | **meaning**
---|---
MSK_ORDER_METHOD_APPMINLOC | Approximate minimum local fill-in ordering is employed.
MSK_ORDER_METHOD_TRY_GRAPHPAR | Always try the graph partitioning based ordering.
MSK_ORDER_METHOD_FORCE_GRAPHPAR | Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.
MSK_ORDER_METHOD_NONE | No ordering is used.

**MSK_IPAR_INTPNT_REGULARIZATION_USE** *(string)*: Controls whether regularization is allowed. ↔

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_INTPNT_SCALING** *(string)*: Controls how the problem is scaled before the interior-point optimizer is used. ↔

Default: MSK_SCALING_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SCALING_FREE</td>
<td>The optimizer chooses the scaling heuristic.</td>
</tr>
<tr>
<td>MSK_SCALING_NONE</td>
<td>No scaling is performed.</td>
</tr>
<tr>
<td>MSK_SCALING_MODERATE</td>
<td>A conservative scaling is performed.</td>
</tr>
<tr>
<td>MSK_SCALING_AGGRESSIVE</td>
<td>A very aggressive scaling is performed.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_INTPNT_SOLVE_FORM** *(string)*: Controls whether the primal or the dual problem is solved. ↔

Default: MSK_SOLVE_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SOLVE_FREE</td>
<td>The optimizer is free to solve either the primal or the dual problem.</td>
</tr>
<tr>
<td>MSK_SOLVE_PRIMAL</td>
<td>The optimizer should solve the primal problem.</td>
</tr>
<tr>
<td>MSK_SOLVE_DUAL</td>
<td>The optimizer should solve the dual problem.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_INTPNT_STARTING_POINT** *(string)*: Starting point used by the interior-point optimizer. ↔

Default: MSK_STARTING_POINT_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_STARTING_POINT_FREE</td>
<td>The starting point is chosen automatically.</td>
</tr>
<tr>
<td>MSK_STARTING_POINT_GUESS</td>
<td>The optimizer guesses a starting point.</td>
</tr>
<tr>
<td>MSK_STARTING_POINT_CONSTANT</td>
<td>The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSK_STARTING_POINT_SATISFY_BOUNDS</td>
<td>The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.</td>
</tr>
</tbody>
</table>

**MSK\_IPAR\_LOG** *(integer)*: Controls the amount of log information. 

The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Default: 10

**MSK\_IPAR\_LOG\_BI** *(integer)*: Controls the amount of output printed by the basis identification procedure.

A higher level implies that more information is logged.

Default: 1

**MSK\_IPAR\_LOG\_BI\_FREQ** *(integer)*: Controls logging frequency of the basis identification

Default: 2500

**MSK\_IPAR\_LOG\_CHECK\_CONVEXITY** *(integer)*: Controls logging in convexity check on quadratic problems.

Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default: 0

**MSK\_IPAR\_LOG\_FEAS\_REPAIR** *(integer)*: Controls the amount of output printed when performing feasibility repair.

Default: 1

**MSK\_IPAR\_LOG\_INFEAS\_ANA** *(integer)*: Controls amount of output printed by the infeasibility analyzer procedures.

A higher level implies that more information is logged.

Default: 1

**MSK\_IPAR\_LOG\_INTPNT** *(integer)*: Controls amount of output printed by the interior-point optimizer.

A higher level implies that more information is logged.

Default: 1

**MSK\_IPAR\_LOG\_MIO** *(integer)*: Controls the log level for the mixed-integer optimizer.
A higher level implies that more information is logged.

Default: 4

**MSK_IPAR_LOG_MIO_FREQ** *(integer)*: Controls how frequent the mixed-integer optimizer prints the log line.  
It will print a line every time `MSK_INT_PARMIO_FREQ` relaxations have been solved.

Range: \([-\infty, ..., \infty]\)

Default: 10

**MSK_IPAR_LOG_ORDER** *(integer)*: If turned on, then factor lines are added to the log.

Default: 1

**MSK_IPAR_LOG_PRESOLVE** *(integer)*: Controls amount of output printed by the presolve procedure.

A higher level implies that more information is logged.

Default: 1

**MSK_IPAR_LOG_RESPONSE** *(integer)*: Controls amount of output printed when response codes are reported.

A higher level implies that more information is logged.

Default: 0

**MSK_IPAR_LOG_SIM** *(integer)*: Controls amount of output printed by the simplex optimizer.

A higher level implies that more information is logged.

Default: 4

**MSK_IPAR_LOG_SIM_FREQ** *(integer)*: Controls simplex optimizer logging frequency.

Default: 1000

**MSK_IPAR_LOG_STORAGE** *(integer)*: When turned on, MOSEK prints messages regarding the storage usage and allocation.

Default: 0

**MSK_IPAR_MIO_BRANCH_DIR** *(string)*: Controls whether the mixed-integer optimizer is branching up or down by default.

Default: `MSK_BRANCH_DIR_FREE`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_BRANCH_DIR_FREE</code></td>
<td>The mixed-integer optimizer decides which branch to choose.</td>
</tr>
<tr>
<td><code>MSK_BRANCH_DIR_UP</code></td>
<td>The mixed-integer optimizer always chooses the up branch first.</td>
</tr>
<tr>
<td><code>MSK_BRANCH_DIR_DOWN</code></td>
<td>The mixed-integer optimizer always chooses the down branch first.</td>
</tr>
<tr>
<td><code>MSK_BRANCH_DIR_NEAR</code></td>
<td>Branch in direction nearest to selected fractional variable.</td>
</tr>
<tr>
<td><code>MSK_BRANCH_DIR_FAR</code></td>
<td>Branch in direction farthest from selected fractional variable.</td>
</tr>
<tr>
<td><code>MSK_BRANCH_DIR_ROOT_LP</code></td>
<td>Chose direction based on root lp value of selected variable.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION** (*string*): If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.

Default: `MSK_OFF`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_CUT_CLIQUE** (*string*): Controls whether clique cuts should be generated.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_CUT_CMIR** (*string*): Controls whether mixed integer rounding cuts should be generated.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_CUT_GMI** (*string*): Controls whether GMI cuts should be generated.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_CUTIMPLIED_BOUND** (*string*): Controls whether implied bound cuts should be generated.

Default: `MSK_OFF`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_CUT_KNAPSACK_COVER** (*string*): Controls whether knapsack cover cuts should be generated.

Default: `MSK_OFF`
MSK_IPAR_MIO_CUT_SELECTION_LEVEL (integer): Controls how aggressively generated cuts are selected to be included in the relaxation.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The optimizer chooses the level of cut selection.</td>
</tr>
<tr>
<td>0</td>
<td>Generated cuts less likely to be added to the relaxation.</td>
</tr>
<tr>
<td>1</td>
<td>Cuts are more aggressively selected to be included in the relaxation.</td>
</tr>
</tbody>
</table>

MSK_IPAR_MIO_FEASPUMP_LEVEL (string): Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The optimizer chooses how the Feasibility Pump is used.</td>
</tr>
<tr>
<td>0</td>
<td>The Feasibility Pump is disabled.</td>
</tr>
<tr>
<td>1</td>
<td>The Feasibility Pump is enabled with an effort to improve solution quality.</td>
</tr>
<tr>
<td>2</td>
<td>The Feasibility Pump is enabled with an effort to reach feasibility early.</td>
</tr>
</tbody>
</table>

MSK_IPAR_MIO_HEURISTIC_LEVEL (integer): Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution.

A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Range: {-∞, ..., ∞}

Default: -1

MSK_IPAR_MIO_MAX_NUM_BRANCHES (integer): Maximum number of branches allowed during the branch and bound search.

A negative value means infinite.

Range: {-∞, ..., ∞}

Default: -1

MSK_IPAR_MIO_MAX_NUM_RELAXS (integer): Maximum number of relaxations allowed during the branch and bound search.

A negative value means infinite.

Range: {-∞, ..., ∞}

Default: GAMS NodLim
**MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_rounds** *(integer)*: Maximum number of cut separation rounds at the root node.

Default: 100

**MSK_IPAR_MIO_MAX_NUM_SOLUTIONS** *(integer)*: The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located.

If this parameter has the value \( n > 0 \), then the mixed-integer optimizer will be terminated when \( n \) feasible solutions have been located.

Range: \(-\infty, ..., \infty\)

Default: -1

**MSK_IPAR_MIO_NODE_OPTIMIZER** *(string)*: Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Default: MSK_OPTIMIZER_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_OPTIMIZER_FREE</td>
<td>The optimizer is chosen automatically.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_INTPNT</td>
<td>The interior-point optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_CONIC</td>
<td>The optimizer for problems having conic constraints.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_PRIMAL_SIMPLEX</td>
<td>The primal simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_DUAL_SIMPLEX</td>
<td>The dual simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_FREE_SIMPLEX</td>
<td>One of the simplex optimizers is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_MIXED_INT</td>
<td>The mixed-integer optimizer.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_NODE_SELECTION** *(string)*: Controls the node selection strategy employed by the mixed-integer optimizer.

Default: MSK_MIO_NODE_SELECTION_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_MIO_NODE_SELECTION_FREE</td>
<td>The optimizer decides the node selection strategy.</td>
</tr>
<tr>
<td>MSK_MIO_NODE_SELECTION_FIRST</td>
<td>The optimizer employs a depth first node selection strategy.</td>
</tr>
<tr>
<td>MSK_MIO_NODE_SELECTION_BEST</td>
<td>The optimizer employs a best bound node selection strategy.</td>
</tr>
<tr>
<td>MSK_MIO_NODE_SELECTION_PSEUDO</td>
<td>The optimizer employs selects the node based on a pseudo cost estimate.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE** *(string)*: Enables or disables perspective reformulation in presolve.

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_MIO_PROBING_LEVEL** *(integer)*: Controls the amount of probing employed by the mixed-integer optimizer in presolve. 

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The optimizer chooses the level of probing employed</td>
</tr>
<tr>
<td>0</td>
<td>Probing is disabled</td>
</tr>
<tr>
<td>1</td>
<td>A low amount of probing is employed</td>
</tr>
<tr>
<td>2</td>
<td>A medium amount of probing is employed</td>
</tr>
<tr>
<td>3</td>
<td>A high amount of probing is employed</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT** *(string)*: Use objective domain propagation. 

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_RINS_MAX_NODES** *(integer)*: Controls the maximum number of nodes allowed in each call to the RINS heuristic. 

The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Range: \{-1, ..., ∞\}

Default: -1

**MSK_IPAR_MIO_ROOT_OPTIMIZER** *(string)*: Controls which optimizer is employed at the root node in the mixed-integer optimizer. 

Default: MSK_OPTIMIZER_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_OPTIMIZER_FREE</td>
<td>The optimizer is chosen automatically.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_INTPNT</td>
<td>The interior-point optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_CONIC</td>
<td>The optimizer for problems having conic constraints.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_PRIMAL_SIMPLEX</td>
<td>The primal simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_DUAL_SIMPLEX</td>
<td>The dual simplex optimizer is used.</td>
</tr>
<tr>
<td>MSK_OPTIMIZER_FREE_SIMPLEX</td>
<td>One of the simplex optimizers is used.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL** *(integer)*: Controls whether presolve can be repeated at root node. 

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The optimizer chooses whether presolve is repeated.</td>
</tr>
</tbody>
</table>
### MSK_IPAR_MIO_SEED (integer)

Sets the random seed used for randomization in the mixed integer optimizer.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never repeat presolve.</td>
</tr>
<tr>
<td>1</td>
<td>Always repeat presolve.</td>
</tr>
</tbody>
</table>

Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default: 42

### MSK_IPAR_MIO_VB_DETECTION_LEVEL (integer)

Controls how much effort is put into detecting variable bounds.

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>The optimizer chooses.</td>
</tr>
<tr>
<td>0</td>
<td>No variable bounds are detected.</td>
</tr>
<tr>
<td>1</td>
<td>Only detect variable bounds that are directly represented in the problem.</td>
</tr>
<tr>
<td>2</td>
<td>Detect variable bounds in probing.</td>
</tr>
</tbody>
</table>

Default: -1

### MSK_IPAR_NUM_THREADS (integer)

Controls the number of threads employed by the optimizer.

If set to 0 the number of threads used will be equal to the number of cores detected on the machine. If using the conic optimizer, the value of this parameter set at first optimization remains constant through the lifetime of the process. MOSEK will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting MSK_IPAR_INTPNT_MULTI_THREAD. For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction.

Default: GAMS Threads

### MSK_IPAR_OPF_WRITE_HEADER (string)

Write a text header with date and MOSEK version in an OPF file.

Default: MSK_ON

### MSK_IPAR_OPF_WRITE_HINTS (string)

Write a hint section with problem dimensions in the beginning of an OPF file.

Default: MSK_ON
**MSK_IPAR_OPF_WRITE_LINE_LENGTH** (*integer*): Aim to keep lines in OPF files not much longer than this.  

Default: 80

**MSK_IPAR_OPF_WRITE_PARAMETERS** (*string*): Write a parameter section in an OPF file.  

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_OPF_WRITE_PROBLEM** (*string*): Write objective, constraints, bounds etc. to an OPF file.  

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_OPF_WRITE_SOLUTIONS** (*string*): Enable inclusion of solutions in the OPF files.  

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_OPF_WRITE_SOL_BAS** (*string*): Whether to include basic solution in OPF files.  

If **MSK_IPAR_OPF_WRITE_SOLUTIONS** is **MSK_ON** and a basic solution is defined, include the basic solution in OPF files.  

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_OPF_WRITE_SOL_ITG** (*string*): Whether to include integer solution in OPF files.  

If **MSK_IPAR_OPF_WRITE_SOLUTIONS** is **MSK_ON** and an integer solution is defined, write the integer solution to OPF files.  

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_OPF_WRITE_SOL_ITR (string):** Whether to include interior solution in OPF files. ✏️

If `MSK_IPAR_OPF_WRITE_SOLUTIONS` is `MSK_ON` and an interior solution is defined, write the interior solution to OPF files.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_OPTIMIZER (string):** The parameter controls which optimizer is used to optimize the task. ✏️

Default: `MSK_OPTIMIZER_FREE`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_OPTIMIZER_FREE</code></td>
<td>The optimizer is chosen automatically.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_INTPNT</code></td>
<td>The interior-point optimizer is used.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_CONIC</code></td>
<td>The optimizer for problems having conic constraints.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_PRIMAL_SIMPLEX</code></td>
<td>The primal simplex optimizer is used.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_DUAL_SIMPLEX</code></td>
<td>The dual simplex optimizer is used.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_FREE_SIMPLEX</code></td>
<td>One of the simplex optimizers is used.</td>
</tr>
<tr>
<td><code>MSK_OPTIMIZER_MIXED_INT</code></td>
<td>The mixed-integer optimizer</td>
</tr>
</tbody>
</table>

**MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL (integer):** Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve. ✏️

A negative value means the parameter value is selected automatically.

Range: \{-∞, ..., ∞\}

Default: -1

**MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES (integer):** Control the maximum number of times the eliminator is tried. ✏️

A negative value implies MOSEK decides.

Range: \{-∞, ..., ∞\}

Default: -1

**MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH (integer):** Controls the linear dependency check, which is potentially computationally expensive. ✏️

Range: \{-∞, ..., ∞\}

Default: 100

**MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH (integer):** Controls the linear dependency check, which is potentially computationally expensive. ✏️

Range: \{-∞, ..., ∞\}

Default: 100

**MSK_IPAR_PRESOLVE_LINDEP_USE (string):** Controls whether the linear constraints are checked for linear dependencies. ✏️

Default: `MSK_ON`
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_PRESOLVE_MAX_NUM_PASS** (*integer*): Control the maximum number of times presolve passes over the problem. 

A negative value implies MOSEK decides.

Range: \(-\infty, ..., \infty\)

Default: \(-1\)

**MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS** (*integer*): Controls the maximum number of reductions performed by the presolve.

The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Range: \(-\infty, ..., \infty\)

Default: \(-1\)

**MSK_IPAR_PRESOLVE_USE** (*string*): Controls whether the presolve is applied to a problem before it is optimized.

Default: **MSK_PRESOLVE_MODE_FREE**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_PRESOLVE_MODE_OFF</td>
<td>The problem is not presolved before it is optimized.</td>
</tr>
<tr>
<td>MSK_PRESOLVE_MODE_ON</td>
<td>The problem is presolved before it is optimized.</td>
</tr>
<tr>
<td>MSK_PRESOLVE_MODE_FREE</td>
<td>It is decided automatically whether to presolve before the problem is optimized.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_PTF_WRITE_TRANSFORM** (*string*): If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default: **MSK_ON**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_BASIS_FACTOR_USE** (*string*): Controls whether an LU factorization of the basis is used in a hot-start.

Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default: **MSK_ON**
### MSK_IPAR_SIM_DEGEN (string): Controls how aggressively degeneration is handled.

Default: MSK_SIM_DEGEN_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SIM_DEGEN_NONE</td>
<td>The simplex optimizer should use no degeneration strategy.</td>
</tr>
<tr>
<td>MSK_SIM_DEGEN_FREE</td>
<td>The simplex optimizer chooses the degeneration strategy.</td>
</tr>
<tr>
<td>MSK_SIM_DEGEN_AGGRESSIVE</td>
<td>The simplex optimizer should use an aggressive degeneration strategy.</td>
</tr>
<tr>
<td>MSK_SIM_DEGEN_Moderate</td>
<td>The simplex optimizer should use a moderate degeneration strategy.</td>
</tr>
<tr>
<td>MSK_SIM_DEGEN_MINIMUM</td>
<td>The simplex optimizer should use a minimum degeneration strategy.</td>
</tr>
</tbody>
</table>

### MSK_IPAR_SIM_DUAL_CRASH (integer): Controls whether crashing is performed in the dual simplex optimizer.

In general if a basis consists of more than \((100 \times \text{MSK_IPAR_SIM_DUAL_CRASH})\) percent fixed variables, then a crash will be performed.

Default: 90

### MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION (integer): Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex.

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: \(\{0, ..., 100\}\)

Default: 50

### MSK_IPAR_SIM_DUAL_SELECTION (string): Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Default: MSK_SIM_SELECTION_FREE

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SIM_SELECTION_FREE</td>
<td>The optimizer chooses the pricing strategy.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_FULL</td>
<td>The optimizer uses full pricing.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_ASE</td>
<td>The optimizer uses approximate steepest-edge pricing.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_DEVEX</td>
<td>The optimizer uses devex steepest-edge pricing.</td>
</tr>
<tr>
<td></td>
<td>If it is not available an approximate steep-edge selection is chosen.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_SE</td>
<td>The optimizer uses steepest-edge selection.</td>
</tr>
<tr>
<td></td>
<td>If it is not available an approximate steep-edge selection is chosen.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_PARTIAL</td>
<td>The optimizer uses a partial selection approach.</td>
</tr>
<tr>
<td></td>
<td>The approach is usually beneficial if the number of variables is much larger than the number of constraints.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_SIM_EXPLOIT_DUPVEC** *(string)*: Controls if the simplex optimizers are allowed to exploit duplicated columns. 

Default: `MSK_SIM_EXPLOIT_DUPVEC_OFF`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_SIM_EXPLOIT_DUPVEC_ON</code></td>
<td>Allow the simplex optimizer to exploit duplicated columns.</td>
</tr>
<tr>
<td><code>MSK_SIM_EXPLOIT_DUPVEC_OFF</code></td>
<td>Disallow the simplex optimizer to exploit duplicated columns.</td>
</tr>
<tr>
<td><code>MSK_SIM_EXPLOIT_DUPVEC_FREE</code></td>
<td>The simplex optimizer can choose freely.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_HOTSTART** *(string)*: Controls the type of hot-start that the simplex optimizer perform.

Default: `MSK_SIM_HOTSTART_FREE`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_SIM_HOTSTART_NONE</code></td>
<td>The simplex optimizer performs a coldstart.</td>
</tr>
<tr>
<td><code>MSK_SIM_HOTSTART_FREE</code></td>
<td>The simplex optimize chooses the hot-start type.</td>
</tr>
<tr>
<td><code>MSK_SIM_HOTSTART_STATUS_KEYS</code></td>
<td>Only the status keys of the constraints and variables are used to choose the type of hot-start.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_HOTSTART_LU** *(string)*: Determines if the simplex optimizer should exploit the initial factorization.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_MAX_ITERATIONS** *(integer)*: Maximum number of iterations that can be used by a simplex optimizer.

Default: `GAMS IterLim`

**MSK_IPAR_SIM_MAX_NUM_SETBACKS** *(integer)*: Controls how many set-backs are allowed within a simplex optimizer.

A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default: 250

**MSK_IPAR_SIM_NON_SINGULAR** *(string)*: Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default: `MSK_ON`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_SIM_PRIMAL_CRASH** *(integer)*: Controls whether crashing is performed in the primal simplex optimizer.

In general if a basis consists of more than \((100 \times \text{MSK_IPAR_SIM_PRIMAL_CRASH})\) percent fixed variables, then a crash will be performed.

Default: 90

**MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION** *(integer)*: Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex.

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: \{0, ..., 100\}

Default: 50

**MSK_IPAR_SIM_PRIMAL_SELECTION** *(string)*: Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default: **MSK_SIM_SELECTION_FREE**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SIM_SELECTION_FREE</td>
<td>The optimizer chooses the pricing strategy.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_FULL</td>
<td>The optimizer uses full pricing.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_ASE</td>
<td>The optimizer uses approximate steepest-edge pricing.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_DEVEX</td>
<td>The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_SE</td>
<td>The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.</td>
</tr>
<tr>
<td>MSK_SIM_SELECTION_PARTIAL</td>
<td>The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_REFORMULATION** *(string)*: Controls if the simplex optimizers are allowed to reformulate the problem.

Default: **MSK_SIM_REFORMULATION_OFF**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_SIM_REFORMULATION_ON</td>
<td>Allow the simplex optimizer to reformulate the problem.</td>
</tr>
<tr>
<td>MSK_SIM_REFORMULATION_OFF</td>
<td>Disallow the simplex optimizer to reformulate the problem.</td>
</tr>
<tr>
<td>MSK_SIM_REFORMULATION_FREE</td>
<td>The simplex optimizer can choose freely.</td>
</tr>
<tr>
<td>MSK_SIM_REFORMULATION_AGGRESSIVE</td>
<td>The simplex optimizer should use an aggressive reformulation strategy.</td>
</tr>
</tbody>
</table>
**MSK_IPAR_SIM_SAVE_LU** *(string)*: Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.  

Default: `MSK_OFF`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_ON</code></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><code>MSK_OFF</code></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_SCALING** *(string)*: Controls how much effort is used in scaling the problem before a simplex optimizer is used.  

Default: `MSK_SCALING_FREE`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_SCALING_FREE</code></td>
<td>The optimizer chooses the scaling heuristic.</td>
</tr>
<tr>
<td><code>MSK_SCALING_NONE</code></td>
<td>No scaling is performed.</td>
</tr>
<tr>
<td><code>MSK_SCALING_MModerate</code></td>
<td>A conservative scaling is performed.</td>
</tr>
<tr>
<td><code>MSK_SCALING_AGGRESSIVE</code></td>
<td>A very aggressive scaling is performed.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_SCALING_METHOD** *(string)*: Controls how the problem is scaled before a simplex optimizer is used.  

Default: `MSK_SCALING_METHOD_POW2`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_SCALING_METHOD_POW2</code></td>
<td>Scales only with power of 2 leaving the mantissa untouched.</td>
</tr>
<tr>
<td><code>MSK_SCALING_METHOD_FREE</code></td>
<td>The optimizer chooses the scaling heuristic.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_SEED** *(integer)*: Sets the random seed used for randomization in the simplex optimizers.  

Range: `{0, ..., 32749}`

Default: 23456

**MSK_IPAR_SIM_SOLVE_FORM** *(string)*: Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.  

Default: `MSK_SOLVE_FREE`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_SOLVE_FREE</code></td>
<td>The optimizer is free to solve either the primal or the dual problem.</td>
</tr>
<tr>
<td><code>MSK_SOLVE_PRIMAL</code></td>
<td>The optimizer should solve the primal problem.</td>
</tr>
<tr>
<td><code>MSK_SOLVE_DUAL</code></td>
<td>The optimizer should solve the dual problem.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_SIM_STABILITY_PRIORITY** *(integer)*: Controls how high priority the numerical stability should be given.  

Range: `{0, ..., 100}`

Default: 50
**MSK_IPAR_SIM_SWITCH_OPTIMIZER** *(string)*: Controls the simplex behavior. 

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default: **MSK_OFF**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_ON</strong></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><strong>MSK_OFF</strong></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_TIMING_LEVEL** *(integer)*: Controls the amount of timing performed inside MOSEK. 

Default: 1

**MSK_IPAR_WRITE_COMPRESSION** *(integer)*: Controls whether the data file is compressed while it is written. 

0 means no compression while higher values mean more compression.

Default: 9

**MSK_IPAR_WRITE_DATA_PARAM** *(string)*: If this option is turned on the parameter settings are written to the data file as parameters.

Default: **MSK_OFF**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_ON</strong></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><strong>MSK_OFF</strong></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_GENERIC_NAMES** *(string)*: Controls whether generic names should be used instead of user-defined names when writing to the data file.

Default: **MSK_ON**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_ON</strong></td>
<td>Switch the option on.</td>
</tr>
<tr>
<td><strong>MSK_OFF</strong></td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE GENERIC NAMES IO** *(integer)*: Index origin used in generic names.

Default: 1

**MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS** *(string)*: Controls if the writer ignores incompatible problem items when writing files.

Default: **MSK_OFF**
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_LP_FULL_OBJ** *(string)*: Write all variables, including the ones with 0-coefficients, in the objective.  

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_LP_LINE_WIDTH** *(integer)*: Maximum width of line in an LP file written by MOSEK.  

Range: \(\{40, \ldots, \infty\}\)  

Default: 80

**MSK_IPAR_WRITE_LP_QUOTED_NAMES** *(string)*: If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.  

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_LP STRICT_FORMAT** *(string)*: Controls whether LP output files satisfy the LP format strictly.  

Default: MSK_OFF

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK_OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_LP_TERMS_PER_LINE** *(integer)*: Maximum number of terms on a single line in an LP file written by MOSEK.  

0 means unlimited.  

Default: 10

**MSK_IPAR_WRITE_MPS_FORMAT** *(string)*: Controls in which format the MPS is written.  

Default: MSK_MPS_FORMAT_FREE
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_MPS_FORMAT STRICT</td>
<td>It is assumed that the input file satisfies the MPS format strictly.</td>
</tr>
<tr>
<td>MSK_MPS_FORMAT RELAXED</td>
<td>It is assumed that the input file satisfies a slightly relaxed version of the MPS format.</td>
</tr>
<tr>
<td>MSK_MPS_FORMAT FREE</td>
<td>It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.</td>
</tr>
<tr>
<td>MSK_MPS_FORMAT CPLEX</td>
<td>The CPLEX compatible version of the MPS format is employed.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_MPS_INT** *(string)*: Controls if marker records are written to the MPS file to indicate whether variables are integer restricted. ↔

Default: MSK_ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK.ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK.OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_PRECISION** *(integer)*: Controls the precision with which double numbers are printed in the MPS data file. ↔

In general it is not worthwhile to use a value higher than 15.

Default: 15

**MSK_IPAR_WRITE_TASK_INC_SOL** *(string)*: Controls whether the solutions are stored in the task file too. ↔

Default: MSK.ON

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK.ON</td>
<td>Switch the option on.</td>
</tr>
<tr>
<td>MSK.OFF</td>
<td>Switch the option off.</td>
</tr>
</tbody>
</table>

**MSK_IPAR_WRITE_XML_MODE** *(string)*: Controls if linear coefficients should be written by row or column when writing in the XML file format. ↔

Default: MSK_WRITE_XML_MODE_ROW

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_WRITE_XML_MODE_ROW</td>
<td>Write in row order.</td>
</tr>
<tr>
<td>MSK_WRITE_XML_MODE_COL</td>
<td>Write in column order.</td>
</tr>
</tbody>
</table>

**MSK_SPAR_DATA_FILE_NAME** *(string)*: If set, problem data is written to this file. File extension determines format. ↔

Synonym: writeprob

**MSK_SPAR_PARAM_READ_FILE_NAME** *(string)*: Modifications to the parameter database is read from this file. ↔
**MSK_SPAR_PARAM_WRITE_FILE_NAME** *(string)*: The parameter database is written to this file.  

**MSK_SPAR_WRITE_LP_GEN_VAR_NAME** *(string)*: Sometimes when an LP file is written additional variables must be inserted.  

They will have the prefix denoted by this parameter.  

Default: "xmskgen"

**QEXTRACTALG** *(integer)*: Switch to choose extraction algorithm for quadratic equations in GAMS interface.  

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Choose automatically.</td>
</tr>
<tr>
<td>1</td>
<td>ThreePass: uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage</td>
</tr>
<tr>
<td>2</td>
<td>DoubleForward: uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.</td>
</tr>
</tbody>
</table>

**SDPCHECKVARS** *(boolean)*: Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present.  

Default: 1

**SDPSOLUFILE** *(string)*: Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints.  

**SOLVEFINAL** *(boolean)*: Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished.  

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not solve the fixed problem.</td>
</tr>
<tr>
<td>1</td>
<td>Solve the fixed problem and return duals.</td>
</tr>
</tbody>
</table>

### 5.27.6 Setting up a GAMS/MOSEK-Link license

To use the GAMS/MOSEK solver with a GAMS/MOSEK-Link license you have save a MOSEK standalone license and point to the MOSEK license file via environment variable `MOSEKLM_LICENSE_FILE`.  

---
5.28 NLPEC

5.28.1 Introduction

The GAMS/NLPEC solver, developed jointly by Michael Ferris of UW-Madison and GAMS Development, solves MPEC and MCP models via reformulation of the complementarity constraints. The resulting sequence of NLP models are parameterized by a scalar $\mu$ and solved by existing NLP solvers. The resulting solutions are used to recover an MPEC or MCP solution.

GAMS/NLPEC serves a number of purposes. In many cases, it is an effective tool for solving MPEC models, the first such tool available within GAMS. It also serves as a way to experiment with the many reformulation strategies proposed for solving MPEC and MCP models. Without something like NLPEC (and a library of models to test with) a comprehensive and thorough test and comparison of the various reformulation strategies would not be possible. To better serve these purposes, NLPEC has an open architecture. The model reformulations are written out as GAMS source for solution via an NLP solver, so it is possible to view this source and modify it if desired.

A brief note about notation is in order. The GAMS keyword positive is used to indicate nonnegative variables. The same holds for nonpositive variables and the GAMS keyword negative.

5.28.2 Usage

GAMS/NLPEC can solve models of two types: MPEC and MCP. If you did not specify NLPEC as the default MPEC or MCP solver, use the following statement in your GAMS model before the solve statement:

```gams
option MPEC=nlpec; { or MCP }
```

You can also make NLPEC the default solver via the command line:

```bash
gams nash MPEC=nlpec MCP=nlpec
```

You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section Options) after reading about the different types of reformulations possible (Section Reformulation). In addition, an understanding of the architecture of NLPEC (Section Open Architecture) will be helpful in understanding how GAMS options are treated. Although NLPEC doesn't use the GAMS options workspace, workfactor, optcr, optca, reslim, iterlim, and domlim directly, it passes these options on in the reformulated model so they are available to the NLP subsolver.

5.28.3 Reformulation

In this section we describe the different ways that the NLPEC solver can reformulate an MPEC as an NLP. The description also applies to MCP models: just consider MCP to be an MPEC with a constant objective. The choice of reformulation, and the subsidiary choices each reformulation entails, are controlled by the options described in the section on Setting the Reformulation Options and referenced throughout this section.

The original MPEC model is given as:

$$\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} f(x, y)$$
subject to the constraints

\[ g(x, y) \leq 0 \]  \hspace{1cm} (2)

and

\[ y \text{ solves } MCP(h(x, \cdot), \mathbf{B}). \]  \hspace{1cm} (3)

In most of the reformulations, the objective function (1) is included in the reformulated model without change. In some cases, it may be augmented with a penalty function. The variables \( x \) are typically called upper level variables (because they are associated with the upper level optimization problem) whereas the variables \( y \) are sometimes termed lower level variables.

The constraints (2) are standard nonlinear programming constraints specified in GAMS in the standard fashion. In particular, these constraints may be less than inequalities as shown above, or equalities or greater than inequalities. The constraints will be unaltered by all our reformulations. These constraints may involve both \( x \) and \( y \), or just \( x \) or just \( y \), or may not be present at all in the problem.

The constraints of interest are the equilibrium constraints (3), where (3) signifies that \( y \in \mathbb{R}^m \) is a solution to the mixed complementarity problem (MCP) defined by the function \( h(x, \cdot) \) and the box \( \mathbf{B} \) containing (possibly infinite) simple bounds on the variables \( y \). A point \( y \) with \( a_i \leq y_i \leq b_i \) solves (3) if for each \( i \) at least one of the following holds:

\[ h_i(x, y) = 0 \]
\[ h_i(x, y) \geq 0, \quad y_i = a_i; \]
\[ h_i(x, y) \leq 0, \quad y_i = b_i. \]  \hspace{1cm} (4)

As a special case of (4), consider the case where \( a = 0 \) and \( b = +\infty \). Since \( y_i \) can never be \( +\infty \) at a solution, (4) simplifies to the nonlinear complementarity problem (NCP):

\[ 0 \leq h_i(x, y), 0 \leq y_i \text{ and } y_i h_i(x, y) = 0, \quad i = 1, \ldots, m \]  \hspace{1cm} (5)

namely that \( h \) and \( y \) are nonnegative vectors with \( h \) perpendicular to \( y \). This motivates our shorthand for (4), the "perp to" symbol \( \perp \):

\[ h_i(x, y) \perp y_i \in [a_i, b_i] \]  \hspace{1cm} (6)

The different ways to force (6) to hold using (smooth) NLP constraints are the basis of the NLPEC solver.

We introduce a simple example now that we will use throughout this document for expositional purposes:

\[ \min_{x_1, x_2, y_1, y_2} \quad x_1 + x_2 \]
\[ \text{subject to} \quad x_1 \leq 0, \quad x_2 \leq 0, \quad y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \]
\[ x_2 + y_2 \perp y_2 \in [-1, 1] \]

This problem has the unique solution \( x_1 = 0, x_2 = -1, y_1 = 0, y_2 = 1 \). Note that \( f(x, y) = x_1 + x_2 \) and \( g(x, y) = x_1^2 + x_2^2 - 1 \) are the objective function and the standard nonlinear programming constraints for this problem. The function \( h(x, y) \) is given by:

\[ h(x, y) = \begin{bmatrix} x_1 - y_1 + y_2 - 1 \\ x_2 + y_2 \end{bmatrix} \]

and

\[ a = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} \infty \\ 1 \end{bmatrix}. \]

This example is written very succinctly in GAMS notation as:
5.28 NLPEC

\$TITLE simple mpec example

variable f, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up = 1;
equations cost, g, h1, h2;
cost.. f =E= x1 + x2;
g.. sarr(x1) + sarr(x2) =L= 1;
h1.. x1 =G= y1 - y2 + 1;
h2.. x2 + y2 =N= 0;
model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min f;

Note that the equation cost is used to define f, the constraint g defines the function g, and h is defined by h1 and h2. The complementarity constraints utilize the standard GAMS convention of specifying the orthogonality relationship between h and y in the model statement. The interpretation of the "." relies on the bounds a and b that are specified using positive, negative, or lo and up keywords in GAMS.

Note that since h2 really specifies a function h2 and not a constraint h2(x,y) = 0, we use the GAMS syntax =N= to ensure this is clear here. Since the relationships satisfied by h1 and h2 are determined by the bounds, =G= could also be replaced by =N= in h1.

In describing the various reformulations for (6), it is convenient to partition the y variables into free F, lower bounded L, upper bounded U and doubly bounded B variables respectively, that is:

B := \{ y = (y_F, y_L, y_U, y_B) : a_L ≤ y_L, y_U ≤ b_U, a_B ≤ y_B ≤ b_B \}.

We will assume (without loss of generality) that a_B < b_B. If a_i = b_i then (6) holds trivially for the index i and we can remove the constraint h_i and its corresponding (fixed) variable y_i from the model. The complementarity condition for variables in y_i ∈ F is simply the equality h_i(x,y) = 0 so these equality constraints are moved directly into the NLP constraints g of the original model as equalities. Thus, NLPEC needs only to treat the singly-bounded variables in L and U and the doubly-bounded variables in B. In the above example, L = \{1\}, U = \{\} and B = \{2\}.

5.28.3.1 Product reformulations

Product reformulations all involve products of y_i with h_i, or products of y_i with some auxiliary or slack variables that are set equal to h_i. The underlying point is that the constraints (3) are entirely equivalent to the following system of equalities and inequalities:

\begin{align*}
  w_L &= h_L(x,y), \quad a_L \leq y_L, \quad w_L \geq 0 \quad \text{and} \quad (y_L - a_L)^T w_L = 0 \\
  v_U &= -h_U(x,y), \quad y_U \leq b_U, \quad v_U \geq 0 \quad \text{and} \quad (b_U - y_U)^T v_U = 0 \\
  w_B - v_B &= h_B(x,y), \quad a_B \leq y_B \leq b_B, \quad w_B \geq 0, \quad v_B \geq 0 \\
  (y_B - a_B)^T w_B &= 0, \quad (b_B - y_B)^T v_B = 0.
\end{align*}

Note that each inner product is a summation of products of nonnegative terms: a slack variable and the difference between a variable and its bound. In each of these products, either the slack variable or its complement must be zero in order to have a solution. Complementarity is forced by the multiplication of these two terms. The above reformulation is specified using option reftype mult.

There are a number of variations on this theme, all of which can be specified via an options file. All of the inner products could be put into the same equation, left as in (7) above, or broken out into individual products (one for each i ∈ L ∪ U, two for each i ∈ B). For example, the complementarity constraints
associated with lower bounded variables involve nonnegativity of \( w_L, y_L \geq a_L \) and either of the following alternatives:

\[
(y_L - a_L)^T w_L = \sum_{i \in L} (y_i - a_i) w_i = 0
\]

or

\[
(y_i - a_i) w_i = 0, \quad i = 1, \ldots, m
\]

These different levels of aggregation are chosen using option aggregate none|partial|full.

Since all of the inner products in (7) involve nonnegative terms, we can set the inner products equal to zero or set them \( \leq 0 \) without changing the feasible set. To choose one or the other, use the option constraint equality|inequality.

As a concrete example, consider the option file

```
reftype mult
aggregate none
constraint inequality
```

applied to the simple example given above. Such an option file generates the nonlinear programming model:

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, \quad w_1 \geq 0, \quad y_1 \geq 0 \\
& \quad w_1 y_1 \leq \mu \\
& \quad w_2 - v_2 = x_2 + y_2, \quad w_2, v_2 \geq 0, \quad y_2 \in [-1, 1] \\
& \quad (y_2 + 1) w_2 \leq \mu, \quad (1 - y_2) v_2 \leq \mu
\end{align*}
\]

By default, a single model is generated with the value \( \mu \) set to 0. There are many examples (e.g. interior point codes, many LP and NLP packages, published results on reformulation approaches to MPEC) that illustrate the value of starting with a "nearly-complementary" solution and pushing the complementarity gap down to zero. For this reason, the inner products in (7) above are always set equal to (or \( \leq \)) a scalar \( \mu \) instead of zero. By default \( \mu \) is zero, but options exist to start \( \mu \) at a positive value (e.g. InitMu 1e-2), to decrease it by a constant factor in a series of looped solves (e.g. NumSolves 4, UpdateFac 0.1), and to solve one last time with a final value for \( \mu \) (e.g. FinalMu 0). If the following lines are added to the option file

```
initmu 1.0 
num solves 4
```

then five consecutive solves of the nonlinear program (8) are performed, the first one using \( \mu = 1 \) and each subsequent solve dividing \( \mu \) by 10 (and starting the NLP solver at the solution of the previous model in this sequence).

As a final example, we use a combination of these options to generate a sequence of nonlinear programs whose solutions attempt to trace out the "central path" favored by interior point and barrier algorithms:

```
reftype mult
constraint equality
initmu 1.0
num solves 4
updatefac 0.1
finalmu 1e-6
```

produces 6 nonlinear programs of the form

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, \quad w_1 \geq 0, \quad y_1 \geq 0 \\
& \quad w_1 y_1 = \mu \\
& \quad w_2 - v_2 = x_2 + y_2, \quad w_2, v_2 \geq 0, \quad y_2 \in [-1, 1] \\
& \quad (y_2 + 1) w_2 = \mu, \quad (y_2 - 1) v_2 = \mu
\end{align*}
\]

for values of \( \mu = 1, 0.1, 0.01, 0.001, 0.0001 \) and \( 1e-6 \).
5.28.3.1.1 Slacks and doubly bounded variables  Slack variables can be used to reduce the number of times a complex nonlinear expression appears in the nonlinear programming model, as was carried out in (7). For a simpler illustrative example the NCP constraints (5) are equivalent to the constraints:

\[ w_i = h_i(x, y), 0 \leq w_i, 0 \leq y_i, \text{ and } y_i w_i = 0, \quad i = 1, \ldots, m \]

This reformulation has an additional equality constraint, and additional variables \( w \), but the expression \( h_i \) only appears once. There are cases when this formulation will be preferable, and the simple option `slack none` controls the use of the \( w \) variables.

When there are doubly bounded variables present, these two slack options work slightly differently. For the `positive` case, the reformulation introduces two nonnegative variables \( w_i \) and \( v_i \) that take on the positive and negative parts of \( h_i \) at the solution as shown in (7). Since this is the default value of the option `slack`, the example (8) shows what ensues to both singly and doubly bounded variables under this setting.

For the case `slack none`, Scholtes proposed a way to use a multiplication to force complementarity that requires no slack variables:

\[ h_i \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, (y_i - a_i)h_i \leq \mu, (y_i - b_i)h_i \leq \mu \]  \hspace{1cm} (9)

Note that unlike the inner products in Section Reformulation, we can expect that one of the inequalities in (9) is unlikely to be binding at a solution (i.e. when \( h_i \) is nonzero). Therefore, we cannot use an equality in this reformulation, and furthermore the products must not be aggregated. Thus, if you use this option, the reformulation automatically enforces the additional options `constraint inequality` and `aggregate none` on the doubly bounded variables, even if the user specifies a conflicting option. Thus the option file

\texttt{reftype mult slack none}

results in the model

\[
\begin{align*}
\min_{x_1, x_2, y_1, y_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\
& \quad (x_1 - y_1 + y_2 - 1)y_1 = \mu \\
& \quad y_2 \in [-1, 1], (y_2 + 1)(x_2 + y_2) \leq \mu, (y_2 - 1)(x_2 + y_2) \leq \mu
\end{align*}
\]

Note that the complementarity constraint associated with \( y_1 \) is an equality (the default) while the constraints associated with \( y_2 \) are inequalities for the reasons outlined above.

In the case of doubly bounded variables, a third option is available for the slack variables, namely `slack one`. In this case, only one slack is introduced, and this slack removes the need to write the function \( h_i \) twice in the reformulated model as follows:

\[ h_i(x, y) \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, w_i = h_i(x, y), (y_i - a_i)w_i \leq \mu, (y_i - b_i)w_i \leq \mu \]

Note that the slack variable \( w \) that is introduced is a free variable. It is not known before solving the problem whether \( w_i \) will be positive or negative at the solution.

We take this opportunity to introduce a simple extension to our option mechanism, namely the ability to set the options for singly and doubly bounded variables differently. For example, the option file

\texttt{reftype mult slack positive one}
sets the option **slack positive** for the singly bounded variables and the option **slack one** for the doubly bounded variables resulting in the model

\[
\begin{align*}
\min & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\
& \quad w_1 y_1 = \mu_1 \\
& \quad w_2 = x_2 + y_2, y_2 \in [-1,1], (y_2 + 1)w_2 \leq \mu_2, (y_2 - 1)w_2 \leq \mu_2
\end{align*}
\]

Additional options such as

- **initmu** 1.0 3.0
- **numsolves** 2
- **updatefac** 0.1 0.2

allow the values of \( \mu \) for the singly and doubly bounded variables to be controlled separately. In this case \( \mu_1 \) takes on values of 1, 0.1 and 0.01, while \( \mu_2 \) takes on values 3.0, 0.6 and 0.12 in each of the three nonlinear programming models generated.

### 5.28.3.2 NCP functions

An NCP-function is a function \( \phi(r, s) \) with the following property:

\[
\phi(r, s) = 0 \iff r \geq 0, s \geq 0, rs = 0
\]

Clearly, finding a zero of an NCP-function solves a complementarity problem in \((r, s)\). We can replace the inner products of nonnegative vectors in (7) with a vector of NCP functions whose arguments are complementary pairs, e.g. \((y_i - a_i, w_i)\) becomes \(\phi(y_i - a_i, w_i) = 0\), \(i \in \mathcal{L}\) and arrive at another way to treat the complementarity conditions. Note that an NCP function forces both nonnegativity and complementarity, so constraints to explicitly force nonnegativity are not required, though they can be included.

Examples of NCP functions include the min function, \(\min(r, s)\), and the Fischer-Burmeister function

\[
\phi(r, s) = \sqrt{r^2 + s^2} - r - s
\]

There is no requirement that an NCP function be nonnegative everywhere (it may be strictly negative at some points), so there is little point in setting the option **constraint**; it will automatically take on the value **constraint equality**. NCP functions cannot be aggregated, so the **aggregate** option will always be set to **none**.

Since the arguments to the NCP functions are going to be nonnegative at solution, we cannot use the functions \(h_i\) directly in the case of doubly-bounded variables. We must use slacks \(w - v = h_i\) to separate \(h_i\) into its positive and negative parts (but see Section **Doubly bounded variables** below). The slacks can be **positive** or **free**, since the NCP function will force positivity at solution. For the singly-bounded variables, slacks are optional, and can also be **positive** or **free**.

Both of the NCP functions mentioned above suffer from being non-differentiable at the origin (and at points where \(r = s\) for the min function). Various smoothed NCP-functions have been proposed that are differentiable. These smooth functions are parameterized by \(\mu\), and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation \(\mu\) that guarantees differentiability:

\[
\phi_{FB}(r, s) := \sqrt{r^2 + s^2} + 2\mu - (r + s).
\]  

(10)

You can choose these particular NCP functions using option **RefType min|FB|fFB**. The difference between the last two is that **RefType FB** writes out GAMS code to compute the function \(\phi_{FB}\), while **RefType fFB** makes use of the GAMS intrinsic function \(\text{NCPF}B(r, s, \mu)\) that computes \(\phi_{FB}\) internally. In general, using the GAMS intrinsic function should work better since the intrinsic can guard against overflow, scale the arguments before computing the function, and use alternative formulas that give more accurate results for certain input ranges.

As an example, the option file
reftype fFB
slack free
initmu 1e-2

generates the reformulation

\[
\begin{align*}
\min_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1 \\
& \quad \phi_{FB}(w_1,y_1,\mu) = 0 \\
& \quad w_2 - v_2 = x_2 + y_2 \\
& \quad \phi_{FB}(y_2 + 1, w_2, \mu) = 0, \quad \phi_{FB}(1 - y_2, v_2, \mu) = 0
\end{align*}
\]

with a value of \( \mu = 0.01 \). Following a path of solutions for decreasing values of \( \mu \) is possible using the options discussed above.

Each of the two arguments to the NCP function will be nonnegative at solution, but for each argument we have the option of including a nonnegativity constraint explicitly as well. This results in the 4 values for the option NCPBounds none|all|function|variable. When no slacks are present, this option controls whether to bound the function \( h_i \) as well as including it in the NCP function, e.g. \( h_i \geq 0, \phi(h_i, y_i - a_i) = 0 \). When slacks are present, we require that the slack setting be consistent with the bound setting for the function argument to the NCP function, where NCPBounds none|variable is consistent with free slack variables and NCPBounds all|function is consistent with positive slack variables.

Thus, the option file

reftype min
slack positive
NCPBounds function

generates the reformulation

\[
\begin{align*}
\min_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0 \\
& \quad \min(w_1,y_1) = \mu \\
& \quad w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0 \\
& \quad \min(y_2 + 1, w_2) = \mu, \min(1 - y_2, v_2) = \mu
\end{align*}
\]

The NCPBounds function option means that the variable argument to the NCP function (in this case \( y \)) does not have its bounds explicitly enforced. It should be noted that this nonlinear program has nondifferentiable constraints for every value of \( \mu \). For this reason, the model is constructed as a dnlp model (instead of an nlp model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

\[
\phi_{CM}(r,s) := r - \mu \log(1 + \exp((r - s)/\mu)).
\] (11)

This function is not symmetric in its two arguments, so \( \phi_{CM}(r,s) \neq \phi_{CM}(s,r) \). For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function \( \phi_{FB} \), \( \phi_{CM} \) is not defined in the limit (i.e. for \( \mu = 0 \)) if you use GAMS code to compute it. However, the GAMS intrinsic NCPCM(r,s,mu) handles this limit case internally. The option RefType CMxf|CMfx|fCMxf|fCMfx chooses a reformulation based on the function \( \phi_{CM} \). Again, the last two choices use the GAMS intrinsic function.
5.28.3.2.1 Doubly bounded variables Like the mult reformulation (7), reformulations using NCP functions are appropriate as long as we split the function $h_i$ matching a doubly-bounded variable into its positive and negative parts $w_i = v_i = h_i$. To avoid this, Billups has proposed using a composition of NCP functions to treat the doubly-bounded case:

$$h_i \perp a_i \leq y_i \leq b_i \iff \phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \quad (12)$$

Use option RefType | fBill to choose such a reformulation for the doubly-bounded variables. The first option value writes out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function NCPFB.

5.28.3.3 Penalty functions

All of the reformulations discussed so far have reformulated the complementarity conditions as constraints. It is also possible to treat these by moving them into the objective function with a penalty parameter $1/\mu$: as $\mu$ goes to zero, the relative weight placed on complementarity increases. Ignoring the NLP constraints, we can rewrite the original MPEC problem as

$$\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} f(x, y) + \frac{1}{\mu} (y_L - a_L)^T w_L + (b_L - y_L)^T v_L + (y_B - a_B)^T w_B + (b_B - y_B)^T v_B$$

subject to the constraints

$$w_L = h_L(x, y), \quad a_L \leq y_L, \quad w_L \geq 0$$
$$v_L = -h_L(x, y), \quad y_L \leq b_L, \quad v_L \geq 0$$
$$w_B - v_B = h_B(x, y), \quad a_B \leq y_B \leq b_B, \quad w_B \geq 0, \quad v_B \geq 0 \quad (14)$$

Choose this treatment using option RefType penalty. The options aggregate and constraint are ignored, since the inner products here are all aggregated and there are no relevant constraints. It is possible to do a similar reformulation without using slacks, so the options slack none | positive can be used in conjunction with this reformulation type.

The following option file shows the use of the penalty reformulation, but also indicates how to use a different reformulation for the singly and doubly bounded variables:

```
reftype penalty mult
slack none *
initmu 1.0
numsolve 2
updatefac 0.1 0.2
```

The "*" value allows the slack option to take on its existing value, in this case positive. Applied to our simple example given above, such an option file generates the nonlinear programming model:

$$\min_{x_1, x_2, y_1, y_2, w_2, v_2} \quad x_1 + x_2 + \frac{1}{\mu_1} y_1 (x_1 - y_1 + y_2 - 1)$$
subject to

$$x_1^2 + x_2^2 \leq 1$$
$$x_1 - y_1 + y_2 - 1 \geq 0, \quad y_1 \geq 0$$
$$w_2 - v_2 = x_2 + y_2, \quad w_2, v_2 \geq 0, \quad y_2 \in [-1, 1]$$
$$(y_2 + 1)w_2 \leq \mu_2, \quad (1 - y_2)v_2 \leq \mu_2$$

The penalty parameter $\mu_1$ is controlled separately from the doubly bounded constraint parameter $\mu_2$. For consistency with other options, the penalty parameter in the objective is $1/\mu$ meaning that as $\mu_1$ tends to zero, the penalty increases. The option initmu has only one value, so both the singly and doubly bounded $\mu$ values are initialized to 1. In the above example, three solves are performed with $\mu_1 = 1, 0.1$ and $0.01$ and $\mu_2 = 1, 0.2$ and $0.04$.  

---

2146 Solver Manuals
5.28.3.4 Testing for complementarity

In some cases a solution to the reformulated model may not satisfy the complementarity constraints of the original MPEC, e.g. if a large penalty parameter is used in the reformulation. It can also happen that the solution tolerances used in the NLP solver allow solutions with small error in the NLP model but large error in the original MPEC. For example if \( x = f(x) = .001 \) then the NLP constraint \( xf(x) = 0 \) may satisfy the NLP feasibility tolerance but it's not so easy to claim that either \( x \) or \( f(x) \) is zero. The NLPEC solver includes a check that the proposed solution does in fact satisfy the complementarity constraints. The complementarity gap is computed using the definition common to all GAMS MCP solvers in computing the \( \text{objval} \) model attribute for an MCP model. The tolerance used for this complementarity gap can be adjusted using the \texttt{testtol} option.

5.28.4 Options

For details on how to create and use an option file, see the introductory chapter on solver usage.

For most GAMS solvers, the use of an options file is discouraged, at least for those unfamiliar with the solver. For NLPEC, however, we expect that most users will want to use an options file from the very beginning. NLPEC is as much a tool for experimentation as it is a solver, and as such use of the options file is encouraged.

Option values can take many different types (e.g. strings, integers, or reals). Perhaps the most important option to remember is one with no value at all: the \texttt{help} option. \texttt{Help} prints a list of the available options, along with their possible values and some helpful text. The options file is read sequentially, so in case an option value is set twice, the latter value takes precedence. However, any consistency checks performed on the options values (e.g. \texttt{RefType fBill} cannot be used with \texttt{aggregate full}) are made after the entire options file is read in, so the order in which different options appear is not important, provided the options are not specified twice.

5.28.4.1 Setting the Reformulation Options

While NLPEC has many options, there is a small set of five options that, taken together, serve to define the type of reformulation used. Listed in order of importance (highest priority items first), these reformulation options are the \texttt{RefType}, \texttt{slack}, \texttt{constraint}, \texttt{aggregate} and \texttt{NCPBounds} options. In some cases, setting the highest-priority option \texttt{RefType} is enough to completely define a reformulation (e.g. \texttt{RefType penalty} in the case of doubly-bounded variables). In most cases though, the lower-priority options play a role in defining or modifying a reformulation. It's useful to consider the reformulation options in priority order when creating option files to define reformulations.

Some of the combinations of the reformulation options don't make sense. For example, the use of an NCP function to force complementarity between its two input arguments requires a separate function for each complementary pair, so setting both \texttt{RefType min} and \texttt{aggregate full} is inconsistent. NLPEC implements consistency checks on the reformulation options using the priority order: Given a consistent setting of the higher priority options, the next-highest priority option is checked and, if necessary, reset to be consistent with the items of higher priority. The end result is a set of consistent options that will result in a working reformulation. NLPEC prints out the pre- and post-checked sets of reformulation options, as well as warning messages about changes made. In case you want to use an option that NLPEC doesn't think is consistent, you can use the \texttt{NoCheck} option: this suppresses the consistency checks.

Each of the reformulation options in the table below takes two values - one for the singly-bounded variables in \( L \cup U \) and another for the doubly-bounded variables in \( B \). If one option value appears, it sets both option values. When setting both option values, use an asterisk "\*" to indicate no change. So for example, an option file

\[
\text{RefType fCMxf} \\
\text{RefType * fBill}
\]

first sets the \texttt{RefType} to \texttt{fCMxf} for all variable types, and then resets the \texttt{RefType} to \texttt{fBill} for doubly-bounded variables.
### 5.28.4.2 Reformulation Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>controls constraint aggregation</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Determines if certain constraints are aggregated or not. E.g., to force $w \geq 0$ and $y \geq 0$ to be complementary we can write either $w^T y \leq 0$ or $w_i^T y_i \leq 0$, for all $i$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none: use no aggregation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial: aggregate doubly-bounded variables separately from others</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full: use maximum aggregation possible</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>controls use of equality/inequality</td>
<td>equality</td>
</tr>
<tr>
<td></td>
<td>Determines if certain constraints are written down using equalities or inequalities. E.g., to force $w \geq 0$ and $y \geq 0$ to be complementary we can write either $w^T y \leq 0$ or $w^T y = 0$. This option only plays a role when bounding a quantity whose sign cannot be both positive and negative and which must be 0 at a solution.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>equality: use $=E=$ constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inequality: use $=L=$ constraints</td>
<td></td>
</tr>
<tr>
<td>NCPBounds</td>
<td>sets explicit bounds on arguments of NCP functions</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Determines which of the two arguments to an NCP function $\Phi(r,s)$ are explicitly constrained to be nonnegative. The explicit constraints are in addition to those imposed by the constraint $\Phi(r,s) = 0$, which implies nonnegativity of $r$ and $s$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none: no explicit bounds</td>
<td></td>
</tr>
<tr>
<td></td>
<td>function: explicit bound on function/slack argument</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable: explicit bound on variable argument</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all: explicit bound on both function and variable arguments</td>
<td></td>
</tr>
<tr>
<td>refType</td>
<td>reformulation type</td>
<td>mult</td>
</tr>
<tr>
<td></td>
<td>Determines the type of reformulation used. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find $x \geq 0$, $f(x) \geq 0$, $x^T f(x) = 0$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mult: inner product $&lt;x,f&gt; = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min: NCP function $\min(x,f)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMxf: Chen-Mangasarian NCP function, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMfx: Chen-Mangasarian NCP function, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fCMxf: Chen-Mangasarian NCP function, intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fCMfx: Chen-Mangasarian NCP function, intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FB: Fischer-Burmeister NCP function, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fFB: Fischer-Burmeister NCP function, intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FB_neg: Fischer-Burmeister NCP function negated, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fFB_neg: Fischer-Burmeister NCP function negated, intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bill: Billups function for doubly-bounded variables, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fBill: Billups function for doubly-bounded variables, intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>penalty: weighted penalization of non-complementarity in objective</td>
<td></td>
</tr>
<tr>
<td></td>
<td>median: median function for doubly-bounded variables, explicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fVUsin: Veelken-Ulbrich NCP function (smoothed min), intrinsic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fVUpow: Veelken-Ulbrich NCP function (smoothed min), intrinsic</td>
<td></td>
</tr>
<tr>
<td>slack</td>
<td>control use of slacks for function values</td>
<td>positive</td>
</tr>
<tr>
<td></td>
<td>Determines if slacks are used to treat the functions $h_i$. For single-bounded variables, we use at most one slack (either free or positive) for each $h_i$. For doubly-bounded variables, we can have no slacks, one slack (necessarily free), or two slacks (either free or positive) for each $h_i$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none: no slacks will be used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>free: free slacks will be used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>positive: non-negative slacks will be used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>one: one free slack will be used for each $h_i$ in the doubly bounded case</td>
<td></td>
</tr>
</tbody>
</table>
5.28.4.3 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>allSolves</td>
<td>do all solves in a loop regardless of previous failure</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>In case multiple (looped) solves are specified, the default is to skip sub-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sequence solves when any solve terminates without getting a solution. Setting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>this flag removes the check and all solves are done, regardless of previous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>failures.</td>
<td></td>
</tr>
<tr>
<td>dotGams</td>
<td>name of gams source file for scalar model</td>
<td>auto</td>
</tr>
<tr>
<td>dumpValid</td>
<td>dump valid reformulation options to a GDX file and exit</td>
<td>0</td>
</tr>
<tr>
<td>equReform</td>
<td>final value of parameter mu</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>If specified, an extra solve is carried out with mu set to this value. Can be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set independently for singly and doubly bounded variables.</td>
<td>none</td>
</tr>
<tr>
<td>finalMu</td>
<td>initial value of parameter mu</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A single solve of the nonlinear program is carried out for this value. Note</td>
<td></td>
</tr>
<tr>
<td></td>
<td>that mu must be positive for some settings of refType, e.g. penalty. Can be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set independently for singly and doubly bounded variables.</td>
<td></td>
</tr>
<tr>
<td>initSLo</td>
<td>lower bound for artificials added to the problem</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([-\infty, \infty])</td>
<td></td>
</tr>
<tr>
<td>initSUP</td>
<td>upper bound for artificials added to the problem</td>
<td>+inf</td>
</tr>
<tr>
<td></td>
<td>Range: ([-\infty, \infty])</td>
<td></td>
</tr>
<tr>
<td>noCheck</td>
<td>do not check consistency of reformulation options</td>
<td>0</td>
</tr>
<tr>
<td>numSolves</td>
<td>number of looped solves</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>This should be set in conjunction with the updateFac option, so that the mu</td>
<td></td>
</tr>
<tr>
<td></td>
<td>values are lowered successively.</td>
<td></td>
</tr>
<tr>
<td>parmFile</td>
<td>extra GAMS options for running scalar model</td>
<td></td>
</tr>
<tr>
<td>subSolver</td>
<td>controls what subsolver to run</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>If this option is not specified, the usual GAMS rules for selecting the solv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>er to run are used.</td>
<td></td>
</tr>
<tr>
<td>subSolverOpt</td>
<td>optfile value to pass to the subsolver</td>
<td>0</td>
</tr>
<tr>
<td>terminate</td>
<td>terminate after generating scalar GAMS source code</td>
<td>0</td>
</tr>
<tr>
<td>testTol</td>
<td>tolerance for complementarity check in MPEC/MCP</td>
<td>1e-005</td>
</tr>
<tr>
<td>updateFac</td>
<td>update factor for mu</td>
<td>1e-1</td>
</tr>
<tr>
<td></td>
<td>The factor that multiplies mu before each of the extra solves triggered by</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the numSolves option.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Can be set independently for singly and doubly bounded variables.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: ([1e^{-280}, 1.0])</td>
<td></td>
</tr>
</tbody>
</table>

5.28.4.4 The Outdated equreform Option

In the early versions of NLPEC the only way to set the reform type was via the equreform option. Each valid equreform value represented a preselected combination of the options from Section Setting the Reformulation Options. This made it difficult to experiment with combinations not preselected, so the options in Section Setting the Reformulation Options were added. Be default, the equreform option has value 0 and is not used. To get the old behavior, set equreform to a positive value - this will force the options in Section Setting the Reformulation Options to be ignored. The general options in Section General Options are used no matter how the reformulation type is selected - via RefType or equreform.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>equreform</td>
<td>Old way to set the type of reformulation used.</td>
<td>0</td>
</tr>
</tbody>
</table>

The values allowed for `equreform` and their implications are given by the following table.

<table>
<thead>
<tr>
<th>equreform</th>
<th>L/U</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;,&gt;_i</td>
<td>= μ</td>
</tr>
<tr>
<td>2</td>
<td>&lt;,&gt;_i</td>
<td>= μ</td>
</tr>
<tr>
<td>3</td>
<td>= μ</td>
<td>bad</td>
</tr>
<tr>
<td>4</td>
<td>&lt;,&gt;_L+U+B</td>
<td>= μ</td>
</tr>
<tr>
<td>5</td>
<td>&lt;,&gt;_L+U+B</td>
<td>= μ</td>
</tr>
<tr>
<td>6</td>
<td>&lt;,&gt;_L+U</td>
<td>= μ</td>
</tr>
<tr>
<td>7</td>
<td>&lt;,&gt;_L+U</td>
<td>&lt;μ</td>
</tr>
<tr>
<td>8</td>
<td>= μ</td>
<td>none</td>
</tr>
<tr>
<td>9</td>
<td>&lt;,&gt;_obj</td>
<td>bnd</td>
</tr>
<tr>
<td>10</td>
<td>&lt;,&gt;_obj</td>
<td>none</td>
</tr>
<tr>
<td>11</td>
<td>&lt;,&gt;_obj</td>
<td>= μ</td>
</tr>
<tr>
<td>12</td>
<td>F-B</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>F-B</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>min</td>
<td>= μ</td>
</tr>
<tr>
<td>15</td>
<td>min</td>
<td>&lt;= μ</td>
</tr>
<tr>
<td>16</td>
<td>C-M(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>17</td>
<td>C-M(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>18</td>
<td>&lt;,__,_L</td>
<td>&lt;μ</td>
</tr>
<tr>
<td>19</td>
<td>&lt;,__,_U</td>
<td>&lt;μ</td>
</tr>
<tr>
<td>20</td>
<td>&lt;,__,_L</td>
<td>&lt;= μ</td>
</tr>
<tr>
<td>21</td>
<td>&lt;,__,_L+U+B</td>
<td>= μ</td>
</tr>
<tr>
<td>22</td>
<td>F-B</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>F-B</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>C-M(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>25</td>
<td>C-M(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>26</td>
<td>NCPF</td>
<td>= 0</td>
</tr>
<tr>
<td>27</td>
<td>NCPF</td>
<td>= 0</td>
</tr>
<tr>
<td>28</td>
<td>NCPF</td>
<td>= 0</td>
</tr>
<tr>
<td>29</td>
<td>NCPF</td>
<td>= 0</td>
</tr>
<tr>
<td>30</td>
<td>NCPM(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>31</td>
<td>NCPM(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>32</td>
<td>NCPM(x,f)</td>
<td>= 0</td>
</tr>
<tr>
<td>33</td>
<td>NCPM(x,f)</td>
<td>= 0</td>
</tr>
</tbody>
</table>

Table 1: Values for `equreform` option.

### 5.28.5 Open Architecture

In this section we describe the architecture of the NLPEC solver, i.e. the way the solver is put together. This should be useful to anybody using NLPEC for experiments or to those wanting to know the details of how NLPEC works.

The foundation for the NLPEC solver is the software library (also used in the GAMS/CONVERT solver) that allows us to write out a scalar GAMS model that is mathematically equivalent to the original, or
to write out selected pieces of such a model. Using this software, NLPEC creates a GAMS NLP model using one of the reformulation strategies from Section Reformulation. This model may contain many new variables and/or equations, but it will surely contain the (non)linear expressions defining the original model as well. Once the scalar model has been created, NLPEC calls GAMS to solve this model, using the current NLP solver. The option parmFile can be used to pass on additional options to this GAMS job. After the model has solved, NLPEC reads the NLP solution, extracts the MPEC solution from it, and passes this MPEC solution back to GAMS as it terminates.

There are a number of advantages to this architecture. First, its openness makes it easy to see exactly what reformulation is being done. The intermediate scalar GAMS NLP model can be made available after the run by either saving the scratch directory (i.e. run with keep=1) or using the dotGams option to select an alternate file name. This intermediate model contains all the details of the reformulation. It can be used for debugging in case things didn’t work out as expected. It is also possible to modify this file to do some quick and dirty experiments with similar reformulation strategies. Another advantage is the variety of NLP solvers that can be plugged in to solve the reformulated model. There is no need to program (and debug) an interface to an NLP package to run experiments with an NLP solver - the existing GAMS link is all that is needed. It is also easy to experiment with non-default solver options that may be more appropriate for reformulated MPEC models or for a particular choice of reformulation.

5.29 Octeract Engine

Octeract Engine is a deterministic solver to find globally optimal solutions for mixed-integer nonlinear programs (MINLPs). Distinguishing highlights of Octeract are its powerful and flexible symbolic reformulation engine and its parallelism capabilities. For more detailed information, we refer to the Octeract Knowledge Base.

5.29.1 Usage

The following statement can be used inside your GAMS program to specify that Octeract should be used:

Option SOLVER = OCTERACT;

The above statement should appear before the Solve statement. If Octeract was specified as the default solver during GAMS installation, the above statement is not necessary.

GAMS/Octeract by default uses CPLEX to solve LP/MIP problems, if licensed, and CLP/CBC otherwise. By setting options LP_SOLVER or MILP_SOLVER, other solvers can be selected:

- If a license for GAMS/CPLEX, GAMS/OsiCPLEX, GAMS/CPLEX-link is available, then CPLEX can be used.
- If a Gurobi license is installed on the machine, then Gurobi can be used.
- If a FICO Xpress license is installed on the machine, then Xpress can be used. A GAMS/XPRESS license is currently not sufficient.
5.29.1.1 Specification of Octeract Options

GAMS/OCTERACT supports the GAMS parameters reslim, iterlim, nodlim, workspace, optcr, and threads. The interpretation of threads differs from its original meaning in the sense that also on systems with hyperthreading enabled only the number of actual processor cores is taken into account. Further, GAMS/OCTERACT is limited to the use of at most 16 cores and distributed parallelization is not available.

Options can be specified by a solver options file. A small example for an octeract.opt file is:

```
LP_SOLVER HIGHS
MILP_SOLVER HIGHS
STARTING_POINT_IPOPT VARIABLE_MID_POINT
```

It causes Octeract to use HiGHS to solve LP or MIP problems and to use a different starting point strategy for calls of Ipopt.

5.29.2 List of Octeract Options

In the following, we give a detailed list of all Octeract options.

5.29.2.1 Octeract Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY_CB</td>
<td>This enables or disables constraint probing for domain reduction.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>BIG_COEFF_TOLERANCE</td>
<td>Sets the maximum acceptable absolute value of a coefficient in the lower bounding problem. If a coefficient is greater than this number, the solver will use other, less effective methods instead of solving the lower bounding linear problem (LP).</td>
<td>1e+09</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>BIG_GAP_TOLERANCE</td>
<td>This option states that if the difference between a node’s upper bound and the parent’s lower bound is smaller than BIG_GAP_TOLERANCE, then solve the relaxation.</td>
<td>1e+60</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>BINARY_REFORMULATION_PROCEDURE</td>
<td>If REFORMULATE_BINARIES is enabled, this option determines which method to use in the reformulation of binary variables.</td>
<td>Quadratic_Concave_Constraint</td>
</tr>
<tr>
<td></td>
<td>Range: Quadratic_Concave_Constraint, Quadratic_Equality_Constraint, Exponential_Reformulation, Continuous_Relaxation.</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>BOUND_VIOLATION_TOLERANCE</td>
<td>Bound violation tolerance is used to confirm whether solutions returned by third-party solvers are indeed feasible, within that tolerance. Relaxing this tolerance can help in problems where finding a feasible solution is difficult, or where the global solution is eliminated due to slight constraint violations. Range: [0, 1]</td>
<td>1e-08</td>
</tr>
<tr>
<td>BQCQP_REFORMULATION_METHOD</td>
<td>Type of reformulation used for BQCQP's. The default option is SSLINEARBQCQP which linearises the BQCQP following Sherali and Smith, “An improved linearization strategy for zero-one quadratic programming problems”, 2007. The CONVEXIFY option tries to convexify the problem in a similar way as the convexification for BQP's. See also the Octeract docu. Range: SSLINEARBQCQP, CONVEXIFY</td>
<td>SSLINEARBQCQP</td>
</tr>
<tr>
<td>BQP_REFORMULATION_METHOD</td>
<td>The BQP_REFORMULATION_METHOD option allows the user to select the reformulation method applied to Binary Quadratic Problems during the solving process. See also the Octeract docu. Range: AUTOMATIC, LINEAR, QUADRATIC, SSLINEAR</td>
<td>AUTOMATIC</td>
</tr>
<tr>
<td>BQP_SPARSITY_TOLERANCE</td>
<td>The maximal sparsity for a Binary Quadratic Problem (BQP) to be reformulated as a linear problem. Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>BRANCHING_STRATEGY</td>
<td>The heuristic for selecting the variable to branch in the branch and bound method when solving a problem. See also the Octeract docu. Range: AUTOMATIC, MOST_VIOLATED_TERM, HYBRID_INTEGER_LEAST_REduced_AXIS, MAX_SEPArATION_DISTANCE, STRONG_BRANCHING</td>
<td>AUTOMATIC</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CALCULATE_LB_LARGE_COEFS</td>
<td>If this option is enabled, the Engine will remove constraints with large coefficients when solving a relaxation to find the lower bound of a problem.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>CBC_LB_MAX_NODES</td>
<td>This sets the maximum number of nodes to explore by CBC when it is used to solve a relaxation to find the lower bound of the problem.</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2000000000}</td>
<td></td>
</tr>
<tr>
<td>CBC_PRESOLVE</td>
<td>This enables or disables the CBC presolver.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>CONSTRAINT_VIOLATION_TOLERANCE</td>
<td>This tolerance determines the acceptable constraint violation when applying domain reduction techniques. Relaxing this tolerance can help in problems where finding a feasible solution is difficult, or where the global solution is eliminated due to slight constraint violations.</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>CONVERGENCE_TOLERANCE</td>
<td>This sets a tolerance to determine whether the algorithm has converged to global optimality. Note that this is used as both absolute and a relative gap tolerance. See also the Oysteract docu.</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>CP_MAX_ITERATIONS</td>
<td>This option sets the limit for the number of iterations performed by Constraint Propagation (CP). See also the Oysteract docu.</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>CP_NUMBER_COMPARISON_TOLERANCE</td>
<td>This is the tolerance for comparison of two values within the Constraint Propagation (CP) algorithm, used in particular for the purpose of updating bounds with tighter ones. See also the Oysteract docu.</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CP_SCALING</td>
<td>This option enables or disables Constraint Propagation (CP) scaling. By default, this option is</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>disabled. Enabling this option may help to avoid numerical issues. It is advisable to turn this</td>
<td></td>
</tr>
<tr>
<td></td>
<td>option on for badly scaled problems, e.g. constraints containing coefficients of widely different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scales, wide range of scales for variable bounds, etc. See also the Octeract docu. Range:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>CP_VOLUME_IMPROVEMENT_FACTOR</td>
<td>This sets the required volume improvement factor for iterations of constraint propagation.</td>
<td>0.999</td>
</tr>
<tr>
<td></td>
<td>Constraint Propagation (CP) will stop if it can’t achieve this improvement at a given iteration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See also the Octeract docu. Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>CUT_POOL_MAX_SIZE</td>
<td>This sets the maximum number of cuts to be kept in the cut pool. Range: {0, ..., ∞}</td>
<td>10000</td>
</tr>
<tr>
<td>FBBT_MAX_ITERATIONS</td>
<td>This option sets the maximum number of splits of a variable’s domain when performing Feasi-</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>bility Based Bounds Tightening (FBBT). See also the Octeract docu. Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>FBBT_TIMEOUT</td>
<td>This option sets the maximum amount of time, in seconds, that should be spent when performing</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Feasibility Based Bounds Tightening (FBBT). See also the Octeract docu. Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>FBBT_TOLERANCE</td>
<td>This option sets the minimum size of the subintervals of the domain of a variable when</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>performing Feasibility Based Bounds Tightening (FBBT). See also the Octeract docu. Range:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>FIRST_FEASIBLE_SOLUTION</td>
<td>To turn on First Feasible Solution, the user should specify in a new line of an options file:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>FORCE_EXPANSION</td>
<td>If this option is enabled and USE_AUTOMATIC_EXPANSION is disabled, the Engine will solve the expanded version of your original problem.</td>
<td>0</td>
</tr>
<tr>
<td>HEUR_CONSTRAINT_PENALTY</td>
<td>This option enables or disables the Constraint Penalty heuristic.</td>
<td>OFF</td>
</tr>
<tr>
<td></td>
<td>Range: OFF, NONLINEAR_CONSTRAINTS, ALL_CONSTRAINTS</td>
<td></td>
</tr>
<tr>
<td>HEUR_CONSTRAINT_PENALTY_COEFF</td>
<td>This sets the penalty coefficient for the Constraint Penalty primal heuristic. Higher values will make the solver prioritise feasibility over optimality. However, at the same time, high values may increase the likelihood of numerical instability.</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>HEUR_CS</td>
<td>This option enables or disables the Constraint Satisfaction (CS) heuristic.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_FIXING</td>
<td>This option enables or disables the Fixing heuristic.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_GUIDED_DIVING</td>
<td>This enables or disables the Guided Diving heuristic.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_LB</td>
<td>This option enables or disables the Lower Bound (LB) heuristic.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_MINLP_DIVING</td>
<td>This enables or disables the Diving heuristic for finding better upper bounds (UBs) for Mixed-Integer Nonlinear Programming (MINLP) problems.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_NL_FEASIBILITY_PUMP</td>
<td>This option enables or disables the Nonlinear (NL) Feasibility Pump heuristic.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_PENALITY</td>
<td>This option enables or disables the Penalty heuristic.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>HEUR_QPDIVING</td>
<td>This option enables or disables the Quadratic Programming (QP) Diving heuristic.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>HEUR_SAP</td>
<td>This option enables or disables the Shift and Propagate (SAP) heuristic. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>HEUR_ZERO</td>
<td>This option enables or disables the Zero heuristic. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>INFINITY</td>
<td>This sets the default bound for unbounded variables. Global optimisation requires all variables to be bounded before the solving process begins. If you do not provide bounds for a variable, its bounds will be automatically set to ±INFINITY. The default value is quite conservative - reducing this number can greatly improve performance. Range: [0, ∞]</td>
<td>1e+07</td>
</tr>
<tr>
<td>INTEGER_REFORMULATION_VAR_RANGE_LIMIT</td>
<td>This option sets the maximum bound difference for integer variables in IQP and IQCQP, for which integer reformulation is to be applied. This is necessary to avoid creating huge problems with poor accuracy and numerical issues. Range: {0, ..., 100000}</td>
<td>5000</td>
</tr>
<tr>
<td>INTEGRALITY_VIOLATION_TOLERANCE</td>
<td>Integrality violation tolerance is the acceptable violation before the solver determines that a variable is no longer an integer. Relaxing this tolerance can prevent integer solutions from being fathomed due to small constraint violations. Range: [0, 1]</td>
<td>0.001</td>
</tr>
<tr>
<td>IPOPT_INITIAL_VALUE</td>
<td>When STARTING_POINT_IPOPT is set to CUSTOM_CONSTANT_VALUE, IPOPT_INITIAL_VALUE determines the value of the initialisation of the primal variables given to IPOPT. See also the Octeract docu. Range: real</td>
<td>1</td>
</tr>
<tr>
<td>IS_PURGE_CUTS_ENABLED</td>
<td>The option allows flagging for purging cuts. This is for cut management purposes. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>LLB_TOLERANCE</td>
<td>This tolerance is used to compare solutions of upper bounding problems and lower bounding problems. These two numbers are typically results of two separate local optimisation problems, potentially using two different third-party solvers. Range: $[0, \infty]$</td>
<td>0.001</td>
</tr>
<tr>
<td>LP_SOLVER</td>
<td>This sets the solver which will be used to solve Linear Programming (LP) problems. Range: IPOPT, OSICLP, OSICBC, CPLEX, GUROBI, XPRESS, HIGHS</td>
<td>OSICLP</td>
</tr>
<tr>
<td>MAX_CB_DEPTH</td>
<td>This sets the maximum depth in the branching tree at which constraint probing should be adding additional constraints. If this is set to 0, depth is unlimited. Range: ${0, \ldots, \infty}$</td>
<td>1</td>
</tr>
<tr>
<td>MAX_SOLVER_ITERATIONS</td>
<td>This sets the maximum number of solver iterations in serial mode (this option is ignored in parallel mode). Range: ${1, \ldots, \infty}$</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>MAX_SOLVER_MEMORY</td>
<td>This option sets the memory limit allowed during the solving process, i.e. if the memory used exceeds this limit the solving process is terminated. Range: ${0, \ldots, \infty}$</td>
<td>GAMS workspace</td>
</tr>
<tr>
<td>MAX_SOLVER_TIME</td>
<td>This sets a timeout for the solver in seconds (elapsed real time). Range: $[1, \infty]$</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>MILP_LB_TIMEOUT</td>
<td>This option sets the timeout (in seconds) for the MIP solver that it is used to solve MILP lower bounding problems at every node of the branch-and-bound tree (e.g., relaxations of MINLPs). The suggestion to change this option is usually triggered when the MILP problems often time out. In this case, you have two options: (i) increase the timeout, or (ii) decrease the timeout. A larger timeout could result in some/all MILP problems being solved to global optimality, which can produce superior bounds. If those problems are really hard to solve, it is often practical to reduce the timeout to 1. Doing so effectively means that we don’t waste too much computing time on sub-problems that can not be solved quickly, and redirect that time to the engine's own branch-and-bound. Range: $[1, \infty]$</td>
<td>5</td>
</tr>
<tr>
<td>MILP_SOLVER</td>
<td>This sets the solver which will be used to solve Mixed-integer Linear Programming (MILP) problems. Range: OSICBC, CPLEX, GUROBI, XPRESS, HIGHS</td>
<td>OSICBC</td>
</tr>
<tr>
<td>MIP_SOLVER</td>
<td>Umbrella option to use the specified MIP solver to solve all types of sub-problems it supports. If set to OSICBC/HIGHS, it will overwrite LP_SOLVER and MILP_SOLVER. If set to CPLEX, it will override LP_SOLVER, MILP_SOLVER, QP_SOLVER, and QCQP_SOLVER. Range: CPLEX, OSICBC, HIGHS</td>
<td>CPLEX if licensed, otherwise OSICBC</td>
</tr>
<tr>
<td>NUM_CORES</td>
<td>This sets the number of cores to be used by the Engine to solve the problem. Range: ${-\infty, \ldots, 16}$</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>NUMBER_COMPARISON_TOLERANCE</td>
<td>This tolerance is used to determine whether two floating point numbers are the same. Its use in the solver is context sensitive, as outward rounding is always used to ensure that solutions are not incorrectly eliminated. Range: [0, 1]</td>
<td>1e-06</td>
</tr>
<tr>
<td>OBBT_MAX_DEPTH</td>
<td>This sets the maximum depth in the branching tree at which Optimisation Based Bounds Tightening (OBBT) should be applied. See also the Octeract docu. Range: (1, \ldots, \infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>OBBT_MAX_ITERATIONS</td>
<td>This sets the maximum number of times that Optimisation Based Bounds Tightening (OBBT) must be applied to each variable. See also the Octeract docu. Range: (1, \ldots, \infty)</td>
<td>1</td>
</tr>
<tr>
<td>OUTPUT_FREQUENCY</td>
<td>The solver will print output every OUTPUT_FREQUENCY iterations. Range: (1, \ldots, \infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>OUTPUT_TIME_FREQUENCY</td>
<td>The solver will print output every OUTPUT_TIME_FREQUENCY seconds. Range: (1, \ldots, \infty)</td>
<td>1</td>
</tr>
<tr>
<td>PRINT_SOLUTION</td>
<td>This sets whether to print the solution object on the screen or not, upon termination of the solver. Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>QCQP_SOLVER</td>
<td>Sets the solver that will be used to solve QCQP sub-problems. Range: CPLEX, IPOPT</td>
<td>IPOPT</td>
</tr>
<tr>
<td>QP_SOLVER</td>
<td>Sets the solver that will be used to solve QP sub-problems. Range: CPLEX, IPOPT</td>
<td>IPOPT</td>
</tr>
<tr>
<td>REFORMULATE_BINARYS</td>
<td>This enables or disables the reformulation of binary variables. In particular, it replaces the binary requirement for a variable by adding a set of equivalent constraints according to the option BINARY_REFORMULATIONPROCEDURE. Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>REFORMULATE_INTEGERS</td>
<td>This enables or disables the reformulation of integer variables as binary. Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>REFORMULATE_INTEGERS_IN_MIQCQP</td>
<td>This enables or disables the re-formulation of integer variables as binary in Integer Quadratic Problems (IQP) and Integer Quadratically Constrained Quadratic Problems (IQCQP). Range: Logarithmic_Sum, Linear_Sum</td>
<td>Logarithmic_Sum</td>
</tr>
<tr>
<td>SOLVE_CONTINUOUS_RELAXATION</td>
<td>Setting this option to true will make the solver solve the problem obtained by relaxing the integer and binary variables of the original problem. This can be useful for large problems with a lot of integer or binary variables. For such problems, finding a (feasible) solution may be easier for their continuous relaxation. Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>STARTING_POINT_IPOPT</td>
<td>Starting Point strategy for Ipopt. See also the Octeract docu. Range: CONSTANT_VALUE_ONE, CUSTOM_CONSTANT_VALUE, VARIABLE_MID_POINT, VARIABLE_MAX_POINT, VARIABLE_MIN_POINT</td>
<td>CONSTANT_VALUE_ONE</td>
</tr>
<tr>
<td>STRONG_BRANCHING_VARIABLE_COUNT</td>
<td>When $\text{BRANCHING_STRATEGY} = \text{STRONG_BRANCHING}$, this option sets the (maximum) number of variables to consider for branching. See also the Octeract docu. Range: ${0, ..., \infty}$</td>
<td>100</td>
</tr>
<tr>
<td>UB_FREQUENCY</td>
<td>This option sets how often the solver should solve local optimisation problems to find upper bounds. Small values for this option can be helpful in problems where finding feasible upper bounds is challenging. Range: ${1, ..., \infty}$</td>
<td>4</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>USE_AUTOMATIC_EXPANSION</td>
<td>This enables or disables the use of expansion for mathematical expressions in your problem. If enabled, an alternative expanded version of your problem would be created where functions like ((x-2)^2) will be replaced by ((x^2-4x+4)). The Engine will then select the most promising model to solve between the original (unexpanded) and alternative (expanded) version.</td>
<td>1</td>
</tr>
<tr>
<td>USE_CONVEXITY_CUTS</td>
<td>This enables or disables the use of convexity cuts.</td>
<td>1</td>
</tr>
<tr>
<td>USE_CP</td>
<td>Whether to use Constraint Propagation. See also the Octeract docu.</td>
<td>1</td>
</tr>
<tr>
<td>USE_FBBT</td>
<td>Whether to use Feasibility Based Bound Tightening. See also the Octeract docu.</td>
<td>1</td>
</tr>
<tr>
<td>USE_MILP_RELAXATION</td>
<td>Whether to use mixed-integer linear relaxations or only continuous linear relaxations. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>USE_NONLINEAR_RELAXATION</td>
<td>To turn on nonlinear relaxation for convex MINLPs, the user should specify in a new line of an options file: Range: auto, true, false</td>
<td>auto</td>
</tr>
<tr>
<td>USE_OBBT</td>
<td>Whether to use Optimisation Based Bound Tightening. See also the Octeract docu.</td>
<td>1</td>
</tr>
<tr>
<td>USE_PROBING</td>
<td>This enables or disables probing techniques for domain reduction. See also the Octeract docu.</td>
<td>1</td>
</tr>
<tr>
<td>USE_REduced_COST</td>
<td>This enables or disables the use of reduced costs to tighten variable domains.</td>
<td>1</td>
</tr>
<tr>
<td>USE_REFORMULATION_LINEARIZATION</td>
<td>This option enables or disables the use of reformulation linearisation techniques to create new, valid constraints. Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>
5.30 ODHCPLEX

5.30.1 Introduction

GAMS/ODHCPLEX is a solver from Optimization Direct Inc. that implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models that uses IBM CPLEX as its underlying solver engine. It is designed for large-scale models which a MIP solver would find intractable: either by it being unable to find feasible solutions at all or; more usually, by being unable to find feasible solutions of adequate quality in the time available to its user.

It is intended for users who are familiar with MIP modelling and have some knowledge of using the GAMS/CPLEX solver. GAMS/ODHCPLEX does not demand expert specialism in this field.

ODHCPLEX can be used in two ways: it is implemented as a stand-alone ODHeuristic engine, which can be used on its own (ODHeuristicMethod=STANDALONE); and also within the CPLEX optimizer, within which it can supply and receive solutions from the main CPLEX caller (ODHeuristicMethod=ODH-CPLEX (default)) thereby accelerating optimization compared with GAMS/CPLEX run on its own.

The ODHeuristic engine has a heuristic method for finding an initial feasible solution that it designed to complement, those of CPLEX. Since its main algorithmic procedure works by improving an incumbent feasible solution, getting an initial one is important and may consume a significant part of its total runtime. When used on its own (i.e. ODHeuristicMethod=STANDALONE), users should experiment to discover whether ODHeuristics' or CPLEX's initial feasible solution methods work best, but within ODH-CPLEX (i.e. ODHeuristicMethod=ODH-CPLEX) both methods are run in parallel and the winner is chosen automatically.

ODHeuristics' principal algorithm works by solving a sequence of sub-models. An innovative aspect of this process is its ability to use the model's symbolic structure to achieve the sub-model decomposition. It does this by analyzing the symbolic names that the user gives to the decision variables and careful specification of how this should be done this is worthwhile. ODHCPLEX can work without this analysis, but it usually takes about twice as much runtime.

### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE_SIMPLIFICATION</td>
<td>Enables or disables standard reformulation rules that are applied to canonicalise the problem. See also the Octeract docu. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>USE_STRUCTURE_DETECTOR</td>
<td>Whether linear variables in objective function that are defined by a linear equation should be replaced if possible. Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### Link Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbinary</td>
<td>Whether .nl file should be written in binary form Range: boolean</td>
</tr>
</tbody>
</table>

## 5.29.2.2 Link Options
5.30.2 Specifying Model Structure

The ODHeuristic method needs to break the model down into sub-models. It can do this in one of three ways:

1. Automatically using its decomposition heuristic;

2. Using information specified by the user in the IndexKey parameter;

3. By simply assigning each variable to a different block (or key); or

4. By using the dot option notation with the option .key.

By default, the program will use information specified by the IndexKey parameter if it is set and its automatic decomposition heuristic otherwise. This may be overridden by the Decomposition parameter. If it is set to 0 (zero), option 3 above is selected. If it is set to 1, its automatic decomposition method is used, and if it is set to 2, option 4 above is selected.

Whilst the automatic decomposition method often works well, there may be an advantage to specifying decomposition through the IndexKey parameter. After performing the decomposition in whatever way, the program analyses the decomposition and displays statistics showing the maximum and minimum number of variables in each key or block and showing a percentage score to the decomposition as a whole. A typical display is of the form:

Variables/key 205.58 (+/-304.79), max/min variables/key 933(32) / 60(113).
There are 227 keys (4149 keys were dropped) with 46872 values.
Decomposition score 13.66%, graph score 2074/3135232.

Other things (such as the distribution of variables in keys and the number of keys) being equal, the smaller the percentage decomposition score, the better the decomposition is and the more effective the program will be.

5.30.2.1 Using the IndexKey parameter

The program needs to associate sets of variables with distinct values of a single index. The user can specify this association with a pattern to which some or all of the variables conform. The pattern is in standard C scanf format (see, for example, Kernighan and Ritchie’s 2nd edition of The C Programming Language, section B1.3 Formatted Input). Currently allowable index values must be non-negative integers, so the pattern must include d. For example, if we have variables x whose first index names starts with a number of letters then an underscore followed by a numeric index value (like x(firstone_1), x(another_1), x(firstone_2), x(another_2), ...) the pattern

\[ x(%*[\text{a-z}]_%d) \]

associates those variables whose name ends in _1 with index value 1, those whose name ends in _2 with index value 2 and so on. The pattern is called an index key referred to by the program as the option parameter IndexKey, for example

\[ \text{IndexKey}=%*[\text{xy}(t_%d) \]
It may, however, be desirable to consider variables whose names are, say, \( x(t_2) \) and \( y(t_2) \), to belong to different index values, i.e. to belong to different groups. One way of doing this is to identify them with separate index keys. These can be supplied to \texttt{IndexKey} as two fields separated by a semi-colon, for example

\[
\text{IndexKey}=x(t_{\%d});y(t_{\%d})
\]

Up to 10 fields can be specified in this way.

On the other hand, it may be desirable to consider such variables as having the same index key and their nomenclature may not permit their identification by a single key field. For example, suppose there are variables \( john(t_1), john(t_2), \ldots, jane(t_1), jane(t_2), \ldots \) and \( johnny(t_1), johnny(t_2), \ldots \) and we want to associate \( john(t_x) \) and \( jane(t_x) \) as belonging to key value \( x \), but want to ignore \( johnny(t_x) \).

Two key fields can be used to do this by

\[
\text{INDEXKEY}=john(t_{\%d});jane(t_{\%d})
\]

By default \( john(t_2) \) and \( jane(t_2) \) would not share the same index value, but if the option parameter \texttt{KeyType}, is set to 1, the heuristic will group them together so as to share the same index.

If \texttt{IndexKey} is not specified, the program uses a default decomposition.

The program divides the model up into parts associated with different values of the \texttt{IndexKey} (if specified), using an integer interval divisor. Initially this is a number not less than 2 and it is increased as the search progresses. When no improved solution is found after a number, \texttt{MaxRepeat}, of attempts with the maximum interval divisor, \texttt{MaxInterDiv}, the program terminates. Default values are provided for \texttt{MaxRepeat} and \texttt{MaxInterDiv}, so these do not have to be specified by the user.

### 5.30.3 Heuristic Parameters

There are a number of other parameters which control the behaviour of the heuristic program. These may be left at their default values or specified on the GAMS/ODHCPLEX option file. In addition, many GAMS/CPLEX options can be supplied in the GAMS/ODHCPLEX option file to tweak the CPLEX behavior, see tables below. Note however that - although they can be technically set - these GAMS/CPLEX options may not have any influence on a GAMS/ODHCPLEX solve. For detailed descriptions of the GAMS/CPLEX options, see Detailed Descriptions of CPLEX Options.

Parameter names and their meanings are summarized in the table below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{cpxpresolve}</td>
<td>Applies CPLEX presolve to full model</td>
<td>0</td>
</tr>
<tr>
<td>\texttt{decompdensity}</td>
<td>Matrix density above which automatic decomposition assigns each variable to a separate key</td>
<td>0.3</td>
</tr>
<tr>
<td>\texttt{decomposition}</td>
<td>Model decomposition method.</td>
<td>-1</td>
</tr>
<tr>
<td>\texttt{deterministic}</td>
<td>Specifies whether the solution improvement heuristic is run in deterministic or opportunistic (i.e. non-deterministic) mode</td>
<td>1</td>
</tr>
<tr>
<td>\texttt{divisor}</td>
<td>Initial divisor for sub models</td>
<td>automatic</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>dynamicsearch</td>
<td>Search strategy for CPLEX caller and sub-model solves</td>
<td>-1</td>
</tr>
<tr>
<td>extracplexlog</td>
<td>Write addition CPLEX output to log</td>
<td>0</td>
</tr>
<tr>
<td>feastol</td>
<td>Feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>firstfeas</td>
<td>Use first feasible heuristic for finding an initial feasible solution</td>
<td>-1</td>
</tr>
<tr>
<td>firstfeascontinue</td>
<td>Whether first feasible heuristic continues when it achieves feasibility</td>
<td>0</td>
</tr>
<tr>
<td>firstfeaseffort</td>
<td>Effort limit on first feasible heuristic</td>
<td>-500</td>
</tr>
<tr>
<td>firstfeashift</td>
<td>First feasible heuristic variable shifting in found solutions</td>
<td>1</td>
</tr>
<tr>
<td>globalbounds</td>
<td>Use of global bounds from CPLEX caller</td>
<td>2</td>
</tr>
<tr>
<td>indexkey</td>
<td>Pattern used to match variable names for grouping into sub-models discussed above</td>
<td></td>
</tr>
<tr>
<td>integeronly</td>
<td>Variables to include in INDEXKEY</td>
<td>-1</td>
</tr>
<tr>
<td>integertol</td>
<td>Integrality tolerance for variable values</td>
<td>1e-5</td>
</tr>
<tr>
<td>interdiv</td>
<td>Initial divisor value</td>
<td>4</td>
</tr>
<tr>
<td>.key</td>
<td>Variable block or key number</td>
<td>0</td>
</tr>
<tr>
<td>keytype</td>
<td>Treatment of multiple INDEXKEYs</td>
<td>0</td>
</tr>
<tr>
<td>maxbacktrack</td>
<td>The maximum number of backtracks permitted in sub-model solves</td>
<td>-1</td>
</tr>
<tr>
<td>maxbound</td>
<td>The largest(smallest) non infinite bound value ODH will accept for upper(lower) bounds</td>
<td>1e+9</td>
</tr>
<tr>
<td>maxinfrepeat</td>
<td>Maximum divisor value when solution is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>maxinterdiv</td>
<td>Maximum divisor value</td>
<td>0</td>
</tr>
<tr>
<td>maxrepeat</td>
<td>Maximum number of sub-model repeat solves for each divisor value</td>
<td>0</td>
</tr>
<tr>
<td>newcallback</td>
<td>CPLEX call-back type used</td>
<td>1</td>
</tr>
<tr>
<td>objtarget</td>
<td>Target objective value</td>
<td>0</td>
</tr>
<tr>
<td>odheuristicmethod</td>
<td>ODHeuristic method section</td>
<td>ODH–CPLEX</td>
</tr>
<tr>
<td>odhfeasopt</td>
<td>Optimization method for sub-models in phaseI</td>
<td>0</td>
</tr>
<tr>
<td>odhpresolve</td>
<td>Indicator for the ODHeuristics engine using a separate presolve within ODH-CPLEX</td>
<td>1</td>
</tr>
<tr>
<td>odhthreads</td>
<td>The number of heuristic threads used by ODH-CPLEX or STAN-DALONE</td>
<td>-1</td>
</tr>
<tr>
<td>odhtimelimit</td>
<td>Elapsed time limit in seconds</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>penalty</td>
<td>The objective function coefficient value for penalties</td>
<td>-1</td>
</tr>
<tr>
<td>penperturb</td>
<td>Perturbation tolerance for penalties coefficients</td>
<td>0</td>
</tr>
<tr>
<td>phase12</td>
<td>Specifies whether to use a phaseI/phaseII method to remove infeasibilities</td>
<td>1</td>
</tr>
<tr>
<td>processorlock</td>
<td>Thread allocation</td>
<td>0</td>
</tr>
<tr>
<td>quickfirstsolve</td>
<td>Accelerate initial CPLEX solve</td>
<td>0</td>
</tr>
</tbody>
</table>
## 5.30 ODHCPLEX

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurse</td>
<td>Recurse using heuristic to solve sub-models when a feasible solution has been obtained</td>
<td>0</td>
</tr>
<tr>
<td>recursedecomp</td>
<td>Recursed model decomposition method</td>
<td>0</td>
</tr>
<tr>
<td>recurseiterlim</td>
<td>Recursed heuristic iteration limit for sub-solves</td>
<td>40</td>
</tr>
<tr>
<td>recurselog</td>
<td>Write thread log files for recursed sub-solves</td>
<td>0</td>
</tr>
<tr>
<td>recurseminterlim</td>
<td>Recursed heuristic minimum iterations before a solution is found in sub-solves</td>
<td>0</td>
</tr>
<tr>
<td>recurseconiterlim</td>
<td>Recursed heuristic sub-solves quit after these iterations if a solution is found</td>
<td>maxint</td>
</tr>
<tr>
<td>rejectinfsol</td>
<td>Reject infeasible solutions to sub-models</td>
<td>2</td>
</tr>
<tr>
<td>relaxsos2</td>
<td>Treatment of SOS2 members</td>
<td>1</td>
</tr>
<tr>
<td>seed</td>
<td>Initial random number seed</td>
<td>1234</td>
</tr>
<tr>
<td>strategy</td>
<td>ODH-Cplex Strategy</td>
<td>1</td>
</tr>
<tr>
<td>submodellimit</td>
<td>Node limit for submodel searches</td>
<td>-1</td>
</tr>
<tr>
<td>suborder</td>
<td>Use of priority order in sub-solves</td>
<td>1</td>
</tr>
<tr>
<td>subpresolve</td>
<td>Use of CPLEX presolve in sub-solves</td>
<td>1</td>
</tr>
<tr>
<td>sub_cpx_threads</td>
<td>Threads available for the solves within ODHeuristic</td>
<td>1</td>
</tr>
<tr>
<td>syncfreq</td>
<td>Thread synchronization frequency in deterministic parallel mode</td>
<td>1</td>
</tr>
<tr>
<td>threadlog</td>
<td>Write thread log files</td>
<td>0</td>
</tr>
<tr>
<td>threadzerosync</td>
<td>Which CPLEX threads to use for synchronization</td>
<td>0</td>
</tr>
<tr>
<td>variableclean</td>
<td>Clean variable values from sub-models</td>
<td>0</td>
</tr>
<tr>
<td>zerotol</td>
<td>Zero tolerance for variable values</td>
<td>1e-9</td>
</tr>
</tbody>
</table>

### 5.30.3.2 GAMS/CPLEX: Preprocessing and General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>advind</td>
<td>advanced basis use</td>
<td>1</td>
</tr>
<tr>
<td>aggfill</td>
<td>aggregator fill parameter</td>
<td>10</td>
</tr>
<tr>
<td>aggind</td>
<td>aggregator on/off</td>
<td>-1</td>
</tr>
<tr>
<td>calcqcpduals</td>
<td>calculate the dual values of a quadratically constrained problem</td>
<td>1</td>
</tr>
<tr>
<td>clocktype</td>
<td>clock type for computation time</td>
<td>2</td>
</tr>
<tr>
<td>coeredind</td>
<td>coefficient reduction on/off</td>
<td>-1</td>
</tr>
<tr>
<td>cpumask</td>
<td>switch and mask to bind threads to processors (Linux only)</td>
<td>auto</td>
</tr>
<tr>
<td>cutoff</td>
<td>GMO cutoff</td>
<td>0.0</td>
</tr>
<tr>
<td>datacheck</td>
<td>controls data consistency checking and modeling assistance</td>
<td>0</td>
</tr>
<tr>
<td>depind</td>
<td>dependency checker on/off</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>dettilim</td>
<td>deterministic time limit</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>domlim</td>
<td>domain violation number</td>
<td>0</td>
</tr>
<tr>
<td>eprelax</td>
<td>relaxation for feasOpt</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>feasoptmode</td>
<td>mode of FeasOpt</td>
<td>0</td>
</tr>
<tr>
<td>fixoptfile</td>
<td>name of option file which is read just before solving the fixed problem</td>
<td></td>
</tr>
<tr>
<td>folding</td>
<td>LP folding will be attempted during the preprocessing phase</td>
<td>-1</td>
</tr>
<tr>
<td>freegamsmodel</td>
<td>preserves memory by dumping the GAMS model instance representation temporarily to disk</td>
<td>0</td>
</tr>
<tr>
<td>indicoptstrict</td>
<td>abort in case of an error in indicator constraint in solver option file</td>
<td>1</td>
</tr>
<tr>
<td>lpmethod</td>
<td>algorithm to be used for LP problems</td>
<td>0</td>
</tr>
<tr>
<td>memoryemphasis</td>
<td>reduces use of memory</td>
<td>0</td>
</tr>
<tr>
<td>multobjdisplay</td>
<td>level of display during multiobjective optimization</td>
<td>1</td>
</tr>
<tr>
<td>names</td>
<td>load GAMS names into Cplex</td>
<td>1</td>
</tr>
<tr>
<td>numericalemphasis</td>
<td>emphasizes precision in numerically unstable or difficult problems</td>
<td>0</td>
</tr>
<tr>
<td>optimalitytarget</td>
<td>type of optimality that Cplex targets</td>
<td>0</td>
</tr>
<tr>
<td>parallelmode</td>
<td>parallel optimization mode</td>
<td>0</td>
</tr>
<tr>
<td>paramdisplay</td>
<td>display the nondefault parameters before optimization</td>
<td>1</td>
</tr>
<tr>
<td>predual</td>
<td>give dual problem to the optimizer</td>
<td>0</td>
</tr>
<tr>
<td>preind</td>
<td>turn presolver on/off</td>
<td>1</td>
</tr>
<tr>
<td>prelinear</td>
<td>linear reduction indicator</td>
<td>1</td>
</tr>
<tr>
<td>prepass</td>
<td>number of presolve applications to perform</td>
<td>-1</td>
</tr>
<tr>
<td>prereform</td>
<td>set presolve reformulations</td>
<td>3</td>
</tr>
<tr>
<td>printoptions</td>
<td>list values of all options to GAMS listing file</td>
<td>0</td>
</tr>
<tr>
<td>qextractalg</td>
<td>quadratic extraction algorithm in GAMS interface</td>
<td>0</td>
</tr>
<tr>
<td>qmethod</td>
<td>algorithm to be used for QP problems</td>
<td>0</td>
</tr>
<tr>
<td>qtolin</td>
<td>linearization of the quadratic terms in the objective function of a QP or MIQP model</td>
<td>-1</td>
</tr>
<tr>
<td>randomseed</td>
<td>sets the random seed differently for diversity of solutions</td>
<td>202009243</td>
</tr>
<tr>
<td>record</td>
<td>Records invocations of Callable Library routines</td>
<td>0</td>
</tr>
<tr>
<td>reduce</td>
<td>primal and dual reduction type</td>
<td>3</td>
</tr>
<tr>
<td>relaxfixedinfeas</td>
<td>accept small infeasibilities in the solve of the fixed problem</td>
<td>0</td>
</tr>
<tr>
<td>relaxpreind</td>
<td>presolve for initial relaxation on/off</td>
<td>-1</td>
</tr>
<tr>
<td>rerun</td>
<td>rerun problem if presolve infeasible or unbounded</td>
<td>yes</td>
</tr>
<tr>
<td>scaind</td>
<td>matrix scaling on/off</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>solutiontype</td>
<td>type of solution (basic or non basic) for an LP or QP</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>global default thread count</td>
<td>0</td>
</tr>
<tr>
<td>tilim</td>
<td>overrides the GAMS ResLim option</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>tryint</td>
<td>GMO tryint</td>
<td>0.0</td>
</tr>
<tr>
<td>tuningdettilim</td>
<td>tuning deterministic time limit per model or suite</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>tuningdisplay</td>
<td>level of information reported by the tuning tool</td>
<td>1</td>
</tr>
<tr>
<td>tuningmeasure</td>
<td>measure for evaluating progress for a suite of models</td>
<td>1</td>
</tr>
<tr>
<td>tuningrepeat</td>
<td>number of times tuning is to be repeated on perturbed versions</td>
<td>1</td>
</tr>
<tr>
<td>tuningtilim</td>
<td>tuning time limit per model or suite</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>usebasis</td>
<td>GMO usebasis</td>
<td>0</td>
</tr>
<tr>
<td>warninglimit</td>
<td>determines how many times warnings of a specific type (datacheck=2) will be displayed</td>
<td>10</td>
</tr>
<tr>
<td>workdir</td>
<td>directory for working files</td>
<td></td>
</tr>
<tr>
<td>workmem</td>
<td>memory available for working storage</td>
<td>2048.0</td>
</tr>
</tbody>
</table>

5.30.3.3 GAMS/CPLEX: Simplex Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflictalg</td>
<td>algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model</td>
<td>0</td>
</tr>
<tr>
<td>conflictdisplay</td>
<td>decides how much information CPLEX reports when the conflict refiner is working</td>
<td>1</td>
</tr>
<tr>
<td>craind</td>
<td>crash strategy (used to obtain starting basis)</td>
<td>1</td>
</tr>
<tr>
<td>dpriind</td>
<td>dual simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>dynamicrows</td>
<td>switch for dynamic management of rows</td>
<td>-1</td>
</tr>
<tr>
<td>epper</td>
<td>perturbation constant</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>netfind</td>
<td>attempt network extraction</td>
<td>2</td>
</tr>
<tr>
<td>netppriind</td>
<td>network simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>perind</td>
<td>force initial perturbation</td>
<td>0</td>
</tr>
<tr>
<td>perlim</td>
<td>number of stalled iterations before perturbation</td>
<td>0</td>
</tr>
<tr>
<td>ppriind</td>
<td>primal simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>pricelim</td>
<td>pricing candidate list</td>
<td>0</td>
</tr>
<tr>
<td>reinv</td>
<td>refactorization frequency</td>
<td>0</td>
</tr>
<tr>
<td>sifting</td>
<td>switch for sifting from simplex optimization</td>
<td>1</td>
</tr>
</tbody>
</table>

5.30.3.4 GAMS/CPLEX: Simplex Limit Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>itlim</td>
<td>iteration limit</td>
<td>2147483647</td>
</tr>
<tr>
<td>netitlim</td>
<td>iteration limit for network simplex</td>
<td>2147483647</td>
</tr>
<tr>
<td>objllim</td>
<td>objective function lower limit</td>
<td>-1.0e+75</td>
</tr>
<tr>
<td>objulim</td>
<td>objective function upper limit</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>singlim</td>
<td>limit on singularity repairs</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.30.3.5 GAMS/CPLEX: Simplex Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epmrk</td>
<td>Markowitz pivot tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>epopt</td>
<td>optimality tolerance</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>eprhs</td>
<td>feasibility tolerance</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>netepopt</td>
<td>optimality tolerance for the network simplex method</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>neteprhs</td>
<td>feasibility tolerance for the network simplex method</td>
<td>1.0e-06</td>
</tr>
</tbody>
</table>

### 5.30.3.6 GAMS/CPLEX: Barrier Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baralg</td>
<td>algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barcolnz</td>
<td>dense column handling</td>
<td>0</td>
</tr>
<tr>
<td>barcrossalg</td>
<td>barrier crossover method</td>
<td>0</td>
</tr>
<tr>
<td>barepcomp</td>
<td>convergence tolerance</td>
<td>1.0e-08</td>
</tr>
<tr>
<td>bargrowth</td>
<td>unbounded face detection</td>
<td>1.0e+12</td>
</tr>
<tr>
<td>baritlim</td>
<td>iteration limit</td>
<td>2147483647</td>
</tr>
<tr>
<td>barmaxcor</td>
<td>maximum correction limit</td>
<td>-1</td>
</tr>
<tr>
<td>barobjrng</td>
<td>maximum objective function</td>
<td>1.0e+20</td>
</tr>
<tr>
<td>barorder</td>
<td>row ordering algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barqcpepcomp</td>
<td>convergence tolerance for the barrier optimizer for QCPs</td>
<td>1.0e-07</td>
</tr>
<tr>
<td>barstartalg</td>
<td>barrier starting point algorithm</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.30.3.7 GAMS/CPLEX: Sifting Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>siftalg</td>
<td>sifting subproblem algorithm</td>
<td>0</td>
</tr>
<tr>
<td>siftitlim</td>
<td>limit on sifting iterations</td>
<td>2147483647</td>
</tr>
</tbody>
</table>
## 5.30.3.8 GAMS/Cplex: MIP Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbinterval</td>
<td>best bound interval</td>
<td>7</td>
</tr>
<tr>
<td>bendersstrategy</td>
<td>Benders decomposition algorithm as a strategy</td>
<td>0</td>
</tr>
<tr>
<td>bndstrenind</td>
<td>bound strengthening</td>
<td>-1</td>
</tr>
<tr>
<td>bqpccuts</td>
<td>boolean quadratic polytope cuts for nonconvex QP or MIQP solved to</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>global optimality</td>
<td></td>
</tr>
<tr>
<td>brdir</td>
<td>set branching direction</td>
<td>0</td>
</tr>
<tr>
<td>bttol</td>
<td>backtracking limit</td>
<td>1.0</td>
</tr>
<tr>
<td>cliques</td>
<td>clique cut generation</td>
<td>0</td>
</tr>
<tr>
<td>covers</td>
<td>cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutlo</td>
<td>lower cutoff for tree search</td>
<td>-1.0e+75</td>
</tr>
<tr>
<td>cuts</td>
<td>default cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutsfactor</td>
<td>cut limit</td>
<td>-1.0</td>
</tr>
<tr>
<td>cutup</td>
<td>upper cutoff for tree search</td>
<td>1.0e+75</td>
</tr>
<tr>
<td>disjcuts</td>
<td>disjunctive cuts generation</td>
<td>0</td>
</tr>
<tr>
<td>divetype</td>
<td>MIP dive strategy</td>
<td>0</td>
</tr>
<tr>
<td>eachcutlim</td>
<td>sets a limit for each type of cut</td>
<td>2100000000</td>
</tr>
<tr>
<td>flowcovers</td>
<td>flow cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>flowpaths</td>
<td>flow path cut generation</td>
<td>0</td>
</tr>
<tr>
<td>fpheur</td>
<td>feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>fraccuts</td>
<td>Gomory fractional cut generation</td>
<td>0</td>
</tr>
<tr>
<td>gubcovers</td>
<td>GUB cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>heurfreq</td>
<td>heuristic frequency</td>
<td>0</td>
</tr>
<tr>
<td>heuristic effort</td>
<td>the effort that CPLEX spends on heuristics during a MIP solve</td>
<td>1.0</td>
</tr>
<tr>
<td>implbd</td>
<td>implied bound cut generation</td>
<td>0</td>
</tr>
<tr>
<td>lazy</td>
<td>Lazy constraints activation</td>
<td>0</td>
</tr>
<tr>
<td>lazyconstraints</td>
<td>Indicator to use lazy constraints</td>
<td>0</td>
</tr>
<tr>
<td>lbheur</td>
<td>local branching heuristic</td>
<td>0</td>
</tr>
<tr>
<td>liftprojects</td>
<td>lift-and-project cuts</td>
<td>0</td>
</tr>
<tr>
<td>localimplied</td>
<td>generation of locally valid implied bound cuts</td>
<td>0</td>
</tr>
<tr>
<td>mcfcuts</td>
<td>multi-commodity flow cut generation</td>
<td>0</td>
</tr>
<tr>
<td>mipemphasis</td>
<td>MIP solution tactics</td>
<td>0</td>
</tr>
<tr>
<td>mipkappastats</td>
<td>MIP kappa computation</td>
<td>0</td>
</tr>
<tr>
<td>mipordind</td>
<td>priority list on/off</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>mipordtype</td>
<td>priority order generation</td>
<td>0</td>
</tr>
<tr>
<td>mipsearch</td>
<td>search strategy for mixed integer programs</td>
<td>0</td>
</tr>
<tr>
<td>mipstart</td>
<td>use mip starting values</td>
<td>0</td>
</tr>
<tr>
<td>miqcpstrat</td>
<td>MIQCP relaxation choice</td>
<td>0</td>
</tr>
<tr>
<td>mircuts</td>
<td>mixed integer rounding cut generation</td>
<td>0</td>
</tr>
<tr>
<td>multimipstart</td>
<td>use multiple mipstarts provided via gdx files</td>
<td></td>
</tr>
<tr>
<td>nodecuts</td>
<td>decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree</td>
<td>0</td>
</tr>
<tr>
<td>nodefileind</td>
<td>node storage file indicator</td>
<td>1</td>
</tr>
<tr>
<td>nodesel</td>
<td>node selection strategy</td>
<td>1</td>
</tr>
<tr>
<td>preslvnd</td>
<td>node presolve selector</td>
<td>0</td>
</tr>
<tr>
<td>probe</td>
<td>perform probing before solving a MIP</td>
<td>0</td>
</tr>
<tr>
<td>qpmakepsdind</td>
<td>adjust MIQP formulation to make the quadratic matrix positive-semi-definite</td>
<td>1</td>
</tr>
<tr>
<td>repeatpresolve</td>
<td>reapply presolve at root after preprocessing</td>
<td>-1</td>
</tr>
<tr>
<td>rinsheur</td>
<td>relaxation induced neighborhood search frequency</td>
<td>0</td>
</tr>
<tr>
<td>rltcuts</td>
<td>Reformulation Linearization Technique (RLT) cuts</td>
<td>0</td>
</tr>
<tr>
<td>solvefinal</td>
<td>switch to solve the problem with fixed discrete variables</td>
<td>1</td>
</tr>
<tr>
<td>sos1reform</td>
<td>automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)</td>
<td>0</td>
</tr>
<tr>
<td>sos2reform</td>
<td>automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)</td>
<td>0</td>
</tr>
<tr>
<td>startalg</td>
<td>MIP starting algorithm</td>
<td>0</td>
</tr>
<tr>
<td>strongcandlim</td>
<td>size of the candidates list for strong branching</td>
<td>10</td>
</tr>
<tr>
<td>strongitlim</td>
<td>limit on iterations per branch for strong branching</td>
<td>0</td>
</tr>
<tr>
<td>subalg</td>
<td>algorithm for subproblems</td>
<td>0</td>
</tr>
<tr>
<td>submipnodelem</td>
<td>limit on number of nodes in an RINS subMIP</td>
<td>500</td>
</tr>
<tr>
<td>submipscale</td>
<td>scale the problem matrix when CPLEX solves a subMIP during MIP optimization</td>
<td>0</td>
</tr>
<tr>
<td>submipstartalg</td>
<td>starting algorithm for a subMIP of a MIP</td>
<td>0</td>
</tr>
<tr>
<td>submipsubalg</td>
<td>algorithm for subproblems of a subMIP of a MIP</td>
<td>0</td>
</tr>
<tr>
<td>symmetry</td>
<td>symmetry breaking cuts</td>
<td>-1</td>
</tr>
<tr>
<td>varsel</td>
<td>variable selection strategy at each node</td>
<td>0</td>
</tr>
<tr>
<td>workeralgorithm</td>
<td>set method for optimizing benders subproblems</td>
<td>0</td>
</tr>
<tr>
<td>zerohalfcuts</td>
<td>zero-half cuts</td>
<td>0</td>
</tr>
</tbody>
</table>

5.30.3.9 GAMS/CPLEX: MIP Limit Options
### 5.30 ODHCPLEX

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aggcutlim</code></td>
<td>aggregation limit for cut generation</td>
<td>3</td>
</tr>
<tr>
<td><code>auxrootthreads</code></td>
<td>number of threads for auxiliary tasks at the root node</td>
<td>0</td>
</tr>
<tr>
<td><code>cutpass</code></td>
<td>maximum number of cutting plane passes</td>
<td>0</td>
</tr>
<tr>
<td><code>fraccand</code></td>
<td>candidate limit for generating Gomory fractional cuts</td>
<td>200</td>
</tr>
<tr>
<td><code>fracpass</code></td>
<td>maximum number of passes for generating Gomory fractional cuts</td>
<td>0</td>
</tr>
<tr>
<td><code>intsollim</code></td>
<td>maximum number of integer solutions</td>
<td>2147483647</td>
</tr>
<tr>
<td><code>nodelim</code></td>
<td>maximum number of nodes to solve</td>
<td>2147483647</td>
</tr>
<tr>
<td><code>polishafterdettim</code></td>
<td>deterministic time before starting to polish a feasible solution</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>polishafterepagap</code></td>
<td>absolute MIP gap before starting to polish a feasible solution</td>
<td>0.0</td>
</tr>
<tr>
<td><code>polishafterepgap</code></td>
<td>relative MIP gap before starting to polish a solution</td>
<td>0.0</td>
</tr>
<tr>
<td><code>polishafterintsol</code></td>
<td>MIP integer solutions to find before starting to polish a feasible solution</td>
<td>2147483647</td>
</tr>
<tr>
<td><code>polishafternode</code></td>
<td>nodes to process before starting to polish a feasible solution</td>
<td>2147483647</td>
</tr>
<tr>
<td><code>polishaftertime</code></td>
<td>time before starting to polish a feasible solution</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>probedettim</code></td>
<td>deterministic time spent probing</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>probetime</code></td>
<td>time spent probing</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>repairtries</code></td>
<td>try to repair infeasible MIP start</td>
<td>0</td>
</tr>
<tr>
<td><code>trelim</code></td>
<td>maximum space in memory for tree</td>
<td>1.0e+75</td>
</tr>
</tbody>
</table>

#### 5.30.3.10 GAMS/CPLEX: MIP Solution Pool Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.divflt</code></td>
<td>solution pool range filter coefficients</td>
<td>0</td>
</tr>
<tr>
<td><code>divfltlo</code></td>
<td>lower bound on diversity</td>
<td>mindouble</td>
</tr>
<tr>
<td><code>divfltup</code></td>
<td>upper bound on diversity</td>
<td>maxdouble</td>
</tr>
<tr>
<td><code>populatelim</code></td>
<td>limit of solutions generated for the solution pool by populate method</td>
<td>20</td>
</tr>
<tr>
<td><code>readflt</code></td>
<td>reads Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td><code>solnpool</code></td>
<td>solution pool file name</td>
<td></td>
</tr>
<tr>
<td><code>solnpoolagap</code></td>
<td>absolute tolerance for the solutions in the solution pool</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>solnpoolcapacity</code></td>
<td>limits of solutions kept in the solution pool</td>
<td>2100000000</td>
</tr>
<tr>
<td><code>solnpoolgap</code></td>
<td>relative tolerance for the solutions in the solution pool</td>
<td>1.0e+75</td>
</tr>
<tr>
<td><code>solnpoolintensity</code></td>
<td>solution pool intensity for ability to produce multiple solutions</td>
<td>0</td>
</tr>
<tr>
<td><code>solnpoolmerge</code></td>
<td>solution pool file name for merged solutions</td>
<td></td>
</tr>
<tr>
<td><code>solnpoolnumsym</code></td>
<td>maximum number of variable symbols when writing merged solutions</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>solnpoolprefix</td>
<td>file name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>solnpoolreplace</td>
<td>strategy for replacing a solution in the solution pool</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.30.3.11 GAMS/CPLEX: MIP Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bendersfeasctol</td>
<td>Tolerance for whether a feasibility cut has been violated in Benders decomposition</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>bendersoptctol</td>
<td>Tolerance for optimality cuts in Benders decomposition</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>epagap</td>
<td>absolute stopping tolerance</td>
<td>1.0e-06</td>
</tr>
<tr>
<td>epgap</td>
<td>relative stopping tolerance</td>
<td>1.0e-04</td>
</tr>
<tr>
<td>epint</td>
<td>integrality tolerance</td>
<td>1.0e-05</td>
</tr>
<tr>
<td>eplin</td>
<td>degree of tolerance used in linearization</td>
<td>0.001</td>
</tr>
<tr>
<td>objdif</td>
<td>overrides GAMS Cheat parameter</td>
<td>0.0</td>
</tr>
<tr>
<td>relobjdif</td>
<td>relative cheat parameter</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### 5.30.3.12 GAMS/CPLEX: Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bardisplay</td>
<td>progress display level</td>
<td>1</td>
</tr>
<tr>
<td>clonelog</td>
<td>enable clone logs</td>
<td>0</td>
</tr>
<tr>
<td>mipdisplay</td>
<td>progress display level</td>
<td>2</td>
</tr>
<tr>
<td>mipinterval</td>
<td>progress display interval</td>
<td>0</td>
</tr>
<tr>
<td>mpslongnum</td>
<td>MPS file format precision of numeric output</td>
<td>1</td>
</tr>
<tr>
<td>netdisplay</td>
<td>network display level</td>
<td>2</td>
</tr>
<tr>
<td>quality</td>
<td>write solution quality statistics</td>
<td>0</td>
</tr>
<tr>
<td>siftdisplay</td>
<td>sifting display level</td>
<td>1</td>
</tr>
<tr>
<td>simdisplay</td>
<td>simplex display level</td>
<td>1</td>
</tr>
<tr>
<td>writeannotation</td>
<td>produce a Cplex annotation file</td>
<td></td>
</tr>
<tr>
<td>writebas</td>
<td>produce a Cplex basis file</td>
<td></td>
</tr>
<tr>
<td>writelst</td>
<td>produce a Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td>writelp</td>
<td>produce a Cplex LP file</td>
<td></td>
</tr>
<tr>
<td>writemps</td>
<td>produce a Cplex MPS file</td>
<td></td>
</tr>
<tr>
<td>writemst</td>
<td>produce a Cplex mst file</td>
<td></td>
</tr>
<tr>
<td>writeord</td>
<td>produce a Cplex ord file</td>
<td></td>
</tr>
</tbody>
</table>
### 5.30.4 Parallel execution using multiple threads

Both ODHHeuristicMethods STANDALONE and ODH-CPLEX can use multiple simultaneous threads. ODH-CPLEX must use separate threads for the main CPLEX solve and for the ODHeuristics engine. The STANDALONE just uses the ODHeuristics engine which may use multiple simultaneous threads. Thus the processing capability of multi-core hardware can be exploited effectively.

GAMS/ODHCPLEX will ignore the GAMS threads parameter and use its own default. The default ODHeuristic method (i.e. ODH-CPLEX) requires multiple threads to work and with the GAMS threads default of 1 this will not work.

Whilst there are good defaults for allocating available threads, it may be worthwhile to give some attention to the allocation of threads between the main CPLEX solver and the ODHeuristics engine for ODH-CPLEX and STANDALONE.

If the option ODHThreads is set to \( n \), \( n \) threads are allocated in total, otherwise the total number of threads allocated for both ODH-CPLEX and STANDALONE is set to the number of physical processors available on the computer. If the ODHThreads option is set to a number greater than the number of available processors, multiple threads will have to share the same processor, which may severely degrade performance.

In general, the more threads allocated to the main CPLEX solver, the faster it will run, and similarly, the more allocated to the ODHeuristics engine, the faster it will run. The best balance depends on the model being solved and whether it is intended to run to optimality or to an optimality gap of (say) 0.05 or 0.1. If the GAMS/Cplex Threads is not set, ODH-CPLEX defaults to allocating a quarter of the threads to the ODHeuristics engine and the remainder to the main CPLEX solve. Otherwise it allocates the specified number of threads to the main CPLEX solver and the remainder to the ODHeuristics engine.

Within the ODHeuristics engine, the principal heuristic algorithm can run in parallel on multiple threads. Each algorithmic thread uses CPLEX to solve sub-models and each such instance of CPLEX can itself run on multiple threads. So some attention needs to be given to the allocation of threads between them. If SUB_CPX_THREADS is not set, the CPLEX solver will use one thread for each available logical processor to solve the sub-models. This means that only one thread will be available for the solution improvement heuristic. If the option SUB_CPX_THREADS is set, then by default the heuristic engine sets its number of algorithmic threads to

\[
\text{number_of_available_processors} / \text{SUB_CPX_THREADS}
\]

where number_of_available_processors is: the number of logical processors for STANDALONE; and for ODH-CPLEX it is this number less those allocated to the main CPLEX solver.

Many Intel and compatible processors support hyperthreading (where this is enabled on the computer and operating system) and if so there will be two logical processors for every physical core. Using them can severely degrade performance, so if they are enabled it is often a good idea to set ODHThreads to the number of physical processors. Note that on machines with a large number of processors (cores), the principal bottleneck for large scale optimization is usually memory access. In practice it is often better to
use only about half of the available cores on (say) a 24 core Intel Xeon system. This is model dependent and some experimentation is worthwhile.

Although the operating system's scheduler usually allocates threads to logical processors so that they run on separate physical cores where possible, it will have more threads to manage than those of the heuristic or CPLEX and so will change this allocation as the heuristic and CPLEX run so as to balance its workload effectively. There is a performance penalty to doing this from the perspective of the heuristic run time. For the ODHeuristics STANDALONE, this can be avoided by locking the heuristic threads to specific processors by setting the heuristic option parameter ProcessorLock to 1. It is not supported for ODH-CPLEX. Under Windows, beware that the threads need to be locked at an above normal priority so this may have a negative impact on other programs concurrently running on the machine.

5.30.5 Determinism

Many users require that repeated runs of their applications under the same conditions give the same results, albeit in slightly variable times. The heuristic runs in this way by default. However, there is a performance penalty that has to be paid for synchronizing the threads. On average, performance can be considerably improved performance at the expense of non-repeatable execution by setting the heuristic option parameter Deterministic to 0. This is often preferred by users with particularly large and difficult models.

5.30.6 Detailed Descriptions of ODHCLPEX Options

advind (integer): advanced basis use

   Default: 1

aggecutlim (integer): aggregation limit for cut generation

   Default: 3

aggfill (integer): aggregator fill parameter

   Default: 10

aggind (integer): aggregator on/off

   Default: -1

auxrootthreads (integer): number of threads for auxiliary tasks at the root node

   Default: 0

baralg (integer): algorithm selection

   Default: 0

barcolnz (integer): dense column handling

   Default: 0

barcrossoalg (integer): barrier crossover method

   Default: 0
**5.30 ODHCPLEX**

`bardisplay` *(integer)*: progress display level

- Default: 1

`barepcomp` *(real)*: convergence tolerance

- Default: $1.0 \times 10^{-8}$

`bargrowth` *(real)*: unbounded face detection

- Default: $1.0 \times 10^{12}$

`baritlim` *(integer)*: iteration limit

- Default: 2147483647

`barmaxcor` *(integer)*: maximum correction limit

- Default: $-1$

`barobjrng` *(real)*: maximum objective function

- Default: $1.0 \times 10^{20}$

`barorder` *(integer)*: row ordering algorithm selection

- Default: 0

`barqcpepcomp` *(real)*: convergence tolerance for the barrier optimizer for QCPs

- Default: $1.0 \times 10^{-7}$

`barstartalg` *(integer)*: barrier starting point algorithm

- Default: 1

`bbinterval` *(integer)*: best bound interval

- Default: 7

`bendersfeascuttol` *(real)*: Tolerance for whether a feasibility cut has been violated in Benders decomposition

- Default: $1.0 \times 10^{-6}$

`bendersoptcuttol` *(real)*: Tolerance for optimality cuts in Benders decomposition

- Default: $1.0 \times 10^{-6}$

`bendersstrategy` *(integer)*: Benders decomposition algorithm as a strategy

- Default: 0

`bndstrenind` *(integer)*: bound strengthening

- Default: $-1$

`bqpcuts` *(integer)*: boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality
Default: 0

**brdir** (integer): set branching direction
Default: 0

**bttol** (real): backtracking limit
Default: 1.0

**calcqcpduals** (integer): calculate the dual values of a quadratically constrained problem
Default: 1

**cliques** (integer): clique cut generation
Default: 0

**clocktype** (integer): clock type for computation time
Default: 2

**clonelog** (integer): enable clone logs
Default: 0

**coeredind** (integer): coefficient reduction on/off
Default: -1

**conflictalg** (integer): algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model
Default: 0

**conflictdisplay** (integer): decides how much information CPLEX reports when the conflict refiner is working
Default: 1

**covers** (integer): cover cut generation
Default: 0

**cpumask** (string): switch and mask to bind threads to processors (Linux only)
Default: auto

**cpxpresolve** (integer): Applies CPLEX presolve to full model
Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not apply</td>
</tr>
<tr>
<td>0</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>1</td>
<td>Always applied</td>
</tr>
</tbody>
</table>
5.30 ODHCPLEX

**craind (integer):** crash strategy (used to obtain starting basis) →

Default: 1

**cutlo (real):** lower cutoff for tree search →

Default: \(-1.0e+75\)

**cutoff (real):** GMO cutoff →

Default: 0.0

**cutpass (integer):** maximum number of cutting plane passes →

Default: 0

**cuts (string):** default cut generation →

Default: 0

**cutsfactor (real):** cut limit →

Default: \(-1.0\)

**cutup (real):** upper cutoff for tree search →

Default: \(1.0e+75\)

**datacheck (integer):** controls data consistency checking and modeling assistance →

Default: 0

**decompdensity (real):** Matrix density above which automatic decomposition assigns each variable to a separate key →

Default: 0.3

**decomposition (integer):** Model decomposition method. →

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>0</td>
<td>Assign each variable to a separate key</td>
</tr>
<tr>
<td>1</td>
<td>Use automatic decomposition method</td>
</tr>
<tr>
<td>2</td>
<td>Use decomposition based on dot option .key</td>
</tr>
</tbody>
</table>

**depind (integer):** dependency checker on/off →

Default: \(-1\)

**deterministic (boolean):** Specifies whether the solution improvement heuristic is run in deterministic or opportunistic (i.e. non-deterministic) mode →

Default: 1
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Opportunistic</td>
</tr>
<tr>
<td>1</td>
<td>Deterministic</td>
</tr>
</tbody>
</table>

**dettilim** *(real)*: deterministic time limit

Default: $1.0e+75$

**disjcuts** *(integer)*: disjunctive cuts generation

Default: 0

**divetype** *(integer)*: MIP dive strategy

Default: 0

**divflt** *(real)*: solution pool range filter coefficients

Default: 0

**divfltol** *(real)*: lower bound on diversity

Default: `mindouble`

**divfltup** *(real)*: upper bound on diversity

Default: `maxdouble`

**divisor** *(integer)*: Initial divisor for sub models

Initial sub model size is model size times 1 over Divisor. Ignored if `InterDiv` is set.

Default: `automatic`

**domlim** *(integer)*: domain violation number

Default: 0

**dprind** *(integer)*: dual simplex pricing

Default: 0

**dynamicrows** *(integer)*: switch for dynamic management of rows

Default: $-1$

**dynamicssearch** *(integer)*: Search strategy for CPLEX caller and sub-model solves

Default: $-1$
**eachcutlim (integer):** sets a limit for each type of cut

Default: 2100000000

**epagap (real):** absolute stopping tolerance

Synonym: optca

Default: $1.0 \times 10^{-6}$

**epgap (real):** relative stopping tolerance

Synonym: optcr

Default: $1.0 \times 10^{-4}$

**epint (real):** integrality tolerance

Default: $1.0 \times 10^{-5}$

**eplin (real):** degree of tolerance used in linearization

Default: 0.001

**epmrk (real):** Markowitz pivot tolerance

Default: 0.01

**epopt (real):** optimality tolerance

Default: $1.0 \times 10^{-6}$

**epper (real):** perturbation constant

Default: $1.0 \times 10^{-6}$

**eprelax (real):** relaxation for feasOpt

Default: $1.0 \times 10^{-6}$

**eprhs (real):** feasibility tolerance

Default: $1.0 \times 10^{-6}$

**extracplexlog (boolean):** Write addition CPLEX output to log

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not write extra CPLEX information</td>
</tr>
<tr>
<td>1</td>
<td>Write extra CPLEX information</td>
</tr>
</tbody>
</table>

**feasoptmode (integer):** mode of FeasOpt

Default: 0
feastol (real): Feasibility tolerance

Range: \([1e-9, 0.1]\)
Default: \(1e-6\)

firstfeas (integer): Use first feasible heuristic for finding an initial feasible solution

The default for ODH-CPLEX is 1 while for the heuristic engine the default is -1.
Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Do not use</td>
</tr>
<tr>
<td>0</td>
<td>Use if no solution found during initial presolve</td>
</tr>
<tr>
<td>1</td>
<td>Always use</td>
</tr>
</tbody>
</table>

firstfeascontinue (boolean): Whether first feasible heuristic continues when it achieves feasibility

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not continue</td>
</tr>
<tr>
<td>1</td>
<td>Continue</td>
</tr>
</tbody>
</table>

firstfeas effort (integer): Effort limit on first feasible heuristic

If the option value is positive the exact value is used as the level of effort. If the value is negative no more than the absolute value of the option is used as the level of effort. The larger the effort level, the more effort is expended before giving up.
Default: -500

firstfeas shift (integer): First feasible heuristic variable shifting in found solutions

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not shift</td>
</tr>
<tr>
<td>1</td>
<td>Moderate shifting</td>
</tr>
<tr>
<td>2</td>
<td>Aggressive shifting</td>
</tr>
</tbody>
</table>

fixoptfile (string): name of option file which is read just before solving the fixed problem

flowcovers (integer): flow cover cut generation
Default: 0

flowpaths (integer): flow path cut generation
Default: 0
**folding (integer):** LP folding will be attempted during the preprocessing phase

Default: -1

**fpheur (integer):** feasibility pump heuristic

Default: 0

**fraccand (integer):** candidate limit for generating Gomory fractional cuts

Default: 200

**fraccuts (integer):** Gomory fractional cut generation

Default: 0

**fracpass (integer):** maximum number of passes for generating Gomory fractional cuts

Default: 0

**freegamsmodel (boolean):** preserves memory by dumping the GAMS model instance representation temporarily to disk

Default: 0

**globalbounds (integer):** Use of global bounds from CPLEX caller

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never use</td>
</tr>
<tr>
<td>2</td>
<td>Always use</td>
</tr>
<tr>
<td>1-4</td>
<td>Intensity of use</td>
</tr>
</tbody>
</table>

**gubcovers (integer):** GUB cover cut generation

Default: 0

**heurfreq (integer):** heuristic frequency

Default: 0

**heuristicEffort (real):** the effort that CPLEX spends on heuristics during a MIP solve

Default: 1.0

**implbd (integer):** implied bound cut generation

Default: 0

**indexkey (string):** Pattern used to match variable names for grouping into sub-models discussed above

**indicoptstrict (boolean):** abort in case of an error in indicator constraint in solver option file

Default: 1

**integeronly (integer):** Variables to include in INDEXKEY

Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>0</td>
<td>All variables</td>
</tr>
<tr>
<td>1</td>
<td>Only non-continuous variables</td>
</tr>
</tbody>
</table>

**integertol** *(real)*: Integrality tolerance for variable values

Default: $1e^{-5}$

**interdiv** *(integer)*: Initial divisor value

Default: 4

**intsollim** *(integer)*: maximum number of integer solutions

Default: 2147483647

**itlim** *(integer)*: iteration limit

Synonym: iterlim

Default: 2147483647

**.key** *(integer)*: Variable block or key number

Default: 0

**keytype** *(boolean)*: Treatment of multiple INDEXKEYs

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Considered separately e.g. INDEXKEY=x₂;y₂ means x₂ and y₂ belong to separate groups</td>
</tr>
<tr>
<td>1</td>
<td>Considered together e.g. INDEXKEY=x₂;y₂ means x₂ and y₂ belong to the same group</td>
</tr>
</tbody>
</table>

**.lazy** *(boolean)*: Lazy constraints activation

Default: 0

**lazyconstraints** *(boolean)*: Indicator to use lazy constraints

Default: 0

**lbheur** *(boolean)*: local branching heuristic

Default: 0

**liftprojcuts** *(integer)*: lift-and-project cuts

Default: 0

**localimplied** *(integer)*: generation of locally valid implied bound cuts
lpmethod (integer): algorithm to be used for LP problems

Default: 0

maxbacktrack (integer): The maximum number of backtracks permitted in sub-model solves

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>0</td>
<td>Infinite</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Use this value if a better solution is available</td>
</tr>
</tbody>
</table>

maxbound (real): The largest (smallest) non infinite bound value ODH will accept for upper (lower) bounds

If this value is positive, bounds exceeding MAXBOUND are reduced to MAXBOUND; if this value is negative, bounds exceeding -MAXBOUND are ignored.

Default: 1e+9

maxinfrepeat (integer): Maximum divisor value when solution is infeasible

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Use this value</td>
</tr>
</tbody>
</table>

maxinterdiv (integer): Maximum divisor value

Default: 0

maxrepeat (integer): Maximum number of sub-model repeat solves for each divisor value

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Use this value</td>
</tr>
</tbody>
</table>

mcfcuts (integer): multi-commodity flow cut generation

Default: 0

memoryemphasis (boolean): reduces use of memory

Default: 0

mipdisplay (integer): progress display level
Default: 2

**mipemphasis** *(integer)*: MIP solution tactics

Default: 0

**mipinterval** *(integer)*: progress display interval

Default: 0

**mipkappastats** *(integer)*: MIP kappa computation

Default: 0

**mipordind** *(boolean)*: priority list on/off

Synonym: prioropt

Default: 1

**mipordtype** *(integer)*: priority order generation

Default: 0

**mipsearch** *(integer)*: search strategy for mixed integer programs

Default: 0

**mipstart** *(integer)*: use mip starting values

Default: 0

**miqcpstrat** *(integer)*: MIQCP relaxation choice

Default: 0

**mircuts** *(integer)*: mixed integer rounding cut generation

Default: 0

**mpslongnum** *(boolean)*: MPS file format precision of numeric output

Default: 1

**multimipstart** *(string)*: use multiple mipstarts provided via gdx files

**multobjdisplay** *(integer)*: level of display during multiobjective optimization

Default: 1

**names** *(boolean)*: load GAMS names into Cplex

Default: 1

**netdisplay** *(integer)*: network display level

Default: 2

**netepopt** *(real)*: optimality tolerance for the network simplex method

Default: 1.0e-06

**neteprhs** *(real)*: feasibility tolerance for the network simplex method

Default: 1.0e-06

**netfind** *(integer)*: attempt network extraction

Default: 2

**netitlim** *(integer)*: iteration limit for network simplex

Default: 2147483647

**netppriind** *(integer)*: network simplex pricing

Default: 0

**newcallback** *(integer)*: CPLEX call-back type used

Default: 1
### nodecuts (integer)

Decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree

- Default: 0

### nodefileind (integer)

Node storage file indicator

- Default: 1

### nodelim (integer)

Maximum number of nodes to solve

- Synonym: nodlim
- Default: 2147483647

### nodesel (integer)

Node selection strategy

- Default: 1

### numericalemphasis (boolean)

Emphasizes precision in numerically unstable or difficult problems

- Default: 0

### objdif (real)

Overrides GAMS Cheat parameter

- Synonym: cheat
- Default: 0.0

### objllim (real)

Objective function lower limit

- Default: -1.0e+75

### objtarget (real)

Target objective value

- ODHeuristics terminates when this value is reached. Defaults to -infinity for minimization or +infinity for maximization models.
- Default: 0

### objulim (real)

Objective function upper limit

- Default: 1.0e+75

### odheuristicmethod (string)

ODHeuristic method section

- Default: ODH-CPLEX
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODH-CPLEX</td>
<td>ODHeuristic within the CPLEX optimizer</td>
</tr>
<tr>
<td>STANDALONE</td>
<td>Stand-alone ODHeuristic engine</td>
</tr>
</tbody>
</table>

**odhfeasopt (boolean):** Optimization method for sub-models in phase 1

Default: 0

**odhpresolve (integer):** Indicator for the ODHeuristics engine using a separate presolve within ODH-CPLEX

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One-step presolve</td>
</tr>
<tr>
<td>2</td>
<td>Two-step presolve</td>
</tr>
</tbody>
</table>

**odhtthreads (integer):** The number of heuristic threads used by ODH-CPLEX or STANDALONE

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>0</td>
<td>Run in serial mode</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Use the specified number of threads</td>
</tr>
</tbody>
</table>

**odhtimelimit (real):** Elapsed time limit in seconds

Synonym: reslim

Default: GAMS ResLim

**optimalitytarget (integer):** type of optimality that Cplex targets

Default: 0

**parallelmode (integer):** parallel optimization mode

Default: 0

**paramdisplay (boolean):** display the nondefault parameters before optimization

Default: 1

**penalty (real):** The objective function coefficient value for penalties

The objective function coefficient value for the penalties introduced to deal with infeasibilities in the solution improvement heuristic. It is set by default when required and if not specified.

Default: -1

**penperturb (real):** Perturbation tolerance for penalties coefficients
Default: 0

perind (boolean): force initial perturbation ←

Default: 0

perlim (integer): number of stalled iterations before perturbation ←

Default: 0

phase12 (boolean): Specifies whether to use a phaseI/phaseII method to remove infeasibilities ←

Default: 1
### value  |  meaning
---|---
0  |  Use composite objective method
1  |  Use phaseI/phaseII method

**polishafterdettim** *(real)*: deterministic time before starting to polish a feasible solution

Default: \(1.0 \times 10^{75}\)

**polishafterepagap** *(real)*: absolute MIP gap before starting to polish a feasible solution

Default: 0.0

**polishafterepgap** *(real)*: relative MIP gap before starting to polish a solution

Default: 0.0

**polishafterintsol** *(integer)*: MIP integer solutions to find before starting to polish a feasible solution

Default: 2147483647

**polishafternode** *(integer)*: nodes to process before starting to polish a feasible solution

Default: 2147483647

**polishaftertime** *(real)*: time before starting to polish a feasible solution

Default: \(1.0 \times 10^{75}\)

**populatelim** *(integer)*: limit of solutions generated for the solution pool by populate method

Default: 20

**ppriind** *(integer)*: primal simplex pricing

Default: 0

**predual** *(integer)*: give dual problem to the optimizer

Default: 0

**preind** *(boolean)*: turn presolver on/off

Default: 1

**prelinear** *(boolean)*: linear reduction indicator

Default: 1

**prepass** *(integer)*: number of presolve applications to perform

Default: -1

**prereform** *(integer)*: set presolve reformulations

Default: 3

**preslvnd** *(integer)*: node presolve selector
5.30 ODHCPLEX

Default: 0

**pricelim (integer):** pricing candidate list

Default: 0

**printoptions (boolean):** list values of all options to GAMS listing file

Default: 0

**probe (integer):** perform probing before solving a MIP

Default: 0

**probedettime (real):** deterministic time spent probing

Default: 1.0e+75

**probetime (real):** time spent probing

Default: 1.0e+75

**processorlock (boolean):** Thread allocation

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not lock threads to processors</td>
</tr>
<tr>
<td>1</td>
<td>Lock threads to processors</td>
</tr>
</tbody>
</table>

**qextractalg (integer):** quadratic extraction algorithm in GAMS interface

Default: 0

**qpmakepsdind (boolean):** adjust MIQP formulation to make the quadratic matrix positive-semi-definite

Default: 1

**qpmethod (integer):** algorithm to be used for QP problems

Default: 0

**qtolin (integer):** linearization of the quadratic terms in the objective function of a QP or MIQP model

Default: -1

**quality (boolean):** write solution quality statistics

Default: 0

**quickfirstsolve (boolean):** Accelerate initial CPLEX solve

Default: 0
**randomseed** (*integer*): sets the random seed differently for diversity of solutions  

Default: 202009243

**readflt** (*string*): reads Cplex solution pool filter file  

**record** (*boolean*): Records invocations of Callable Library routines  

Default: 0

**recurse** (*integer*): Recurse using heuristic to solve sub-models when a feasible solution has been obtained  

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not recurse</td>
</tr>
<tr>
<td>1</td>
<td>Recurse thread 0 only</td>
</tr>
<tr>
<td>2</td>
<td>Recurse odd numbered threads</td>
</tr>
<tr>
<td>3</td>
<td>Recurse all threads</td>
</tr>
<tr>
<td>&lt;0</td>
<td>Recurse when working with an infeasible solution. values are negated (e.g. -3 = recursion of all threads)</td>
</tr>
</tbody>
</table>

**recursedecomp** (*integer*): Recursed model decomposition method  

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Use initial model decomposition</td>
</tr>
<tr>
<td>0</td>
<td>Assign each variable to a separate key</td>
</tr>
<tr>
<td>1</td>
<td>Use automatic decomposition method</td>
</tr>
</tbody>
</table>

**recurseiterlim** (*integer*): Recursed heuristic iteration limit for sub-solves  

Default: 40

**recurselog** (*boolean*): Write thread log files for recursed sub-solves  

Default: 0

**recurseminiterlim** (*integer*): Recursed heuristic minimum iterations before a solution is found in sub-solves  

Default: 0

**recursesoliterlim** (*integer*): Recursed heuristic sub-solves quit after these iterations if a solution is found  

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not recurse</td>
</tr>
<tr>
<td>1</td>
<td>Use existing presolved model</td>
</tr>
</tbody>
</table>
Default: maxint

reduce (integer): primal and dual reduction type ←
   Default: 3

reinv (integer): refactorization frequency ←
   Default: 0

rejectinfsol (integer): Reject infeasible solutions to sub-models ←
   Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not check feasibility or reject</td>
</tr>
<tr>
<td>1</td>
<td>Check feasibility and warn if infeasible, but accept</td>
</tr>
<tr>
<td>2</td>
<td>Check feasibility and reject if infeasible</td>
</tr>
</tbody>
</table>

relaxfixedinfeas (boolean): accept small infeasibilities in the solve of the fixed problem ←
   Default: 0

relaxpreind (integer): presolve for initial relaxation on/off ←
   Default: -1

relaxsos2 (integer): Treatment of SOS2 members ←
   Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Aggressive use in reducing sub-model size</td>
</tr>
<tr>
<td>1</td>
<td>Moderate use in reducing sub-model size</td>
</tr>
<tr>
<td>2</td>
<td>Ignored in sub-model creation</td>
</tr>
</tbody>
</table>

relobjdif (real): relative cheat parameter ←
   Default: 0.0

repairtries (integer): try to repair infeasible MIP start ←
   Default: 0

repeatpresolve (integer): reapply presolve at root after preprocessing ←
   Default: -1

rerun (string): rerun problem if presolve infeasible or unbounded ←
   Default: yes

rinsheur (integer): relaxation induced neighborhood search frequency ←
Default: 0

**rltcuts (integer):** Reformulation Linearization Technique (RLT) cuts

Default: 0

**scaind (integer):** matrix scaling on/off

Default: 0

**seed (integer):** Initial random number seed

Default: 1234

**siftalg (integer):** sifting subproblem algorithm

Default: 0

**siftdisplay (integer):** sifting display level

Default: 1

**sifting (boolean):** switch for sifting from simplex optimization

Default: 1

**siftitlim (integer):** limit on sifting iterations

Default: 2147483647

**simdisplay (integer):** simplex display level

Default: 1

**singlim (integer):** limit on singularity repairs

Default: 10

**solnpool (string):** solution pool file name

**solnpoolagap (real):** absolute tolerance for the solutions in the solution pool

Default: 1.0e+75

**solnpoolcapacity (integer):** limits of solutions kept in the solution pool

Default: 2100000000

**solnpoolgap (real):** relative tolerance for the solutions in the solution pool

Default: 1.0e+75

**solnpoolintensity (integer):** solution pool intensity for ability to produce multiple solutions

Default: 0

**solnpoolmerge (string):** solution pool file name for merged solutions
solnpoolnumsym \(\text{(integer)}\): maximum number of variable symbols when writing merged solutions

Default: 10

solnpoolprefix \(\text{(string)}\): file name prefix for GDX solution files

Default: soin

solnpoolreplace \(\text{(integer)}\): strategy for replacing a solution in the solution pool

Default: 0

solutiontype \(\text{(integer)}\): type of solution (basic or non basic) for an LP or QP

Default: 0

solvefinal \(\text{(boolean)}\): switch to solve the problem with fixed discrete variables

Default: 1

sos1reform \(\text{(integer)}\): automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)

Default: 0

sos2reform \(\text{(integer)}\): automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)

Default: 0

startalg \(\text{(integer)}\): MIP starting algorithm

Default: 0

strategy \(\text{(integer)}\): ODH-Cplex Strategy

The aggressive setting attempt to make more progress with each sub-model solve at the cost of more expensive sub solves. Amongst other changes, it sets InterDiv, MaxInterDiv and MaxRepeat if they are not explicitly set by the user.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Conservative</td>
</tr>
<tr>
<td>1</td>
<td>Normal</td>
</tr>
<tr>
<td>2</td>
<td>Aggressiv</td>
</tr>
</tbody>
</table>

strongcandlim \(\text{(integer)}\): size of the candidates list for strong branching

Default: 10

strongitlim \(\text{(integer)}\): limit on iterations per branch for strong branching

Default: 0

subalg \(\text{(integer)}\): algorithm for subproblems
Default: 0

**submipnode** (integer): limit on number of nodes in an RINS subMIP ↪

Default: 500

**submipscale** (integer): scale the problem matrix when CPLEX solves a subMIP during MIP optimization ↪

Default: 0

**submipstartalg** (integer): starting algorithm for a subMIP of a MIP ↪

Default: 0

**submipsubalg** (integer): algorithm for subproblems of a subMIP of a MIP ↪

Default: 0

**subnode** (integer): Node limit for submodel searches ↪

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Set node limit to this value</td>
</tr>
</tbody>
</table>

**suborder** (integer): Use of priority order in sub-solves ↪

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use any supplied priority order information in sub-solves</td>
</tr>
<tr>
<td>1</td>
<td>Use any supplied priority order information in sub-solves</td>
</tr>
</tbody>
</table>

**subpresolve** (integer): Use of CPLEX presolve in sub-solves ↪

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use normal presolve</td>
</tr>
<tr>
<td>2</td>
<td>Use two-step presolve</td>
</tr>
</tbody>
</table>

**sub_cpx_threads** (integer): Threads available for the solves within ODHeuristic ↪

Default: 1

**symmetry** (integer): symmetry breaking cuts ↪

Default: -1

**syncfreq** (integer): Thread synchronization frequency in deterministic parallel mode ↪

Default: 1
### threadlog (boolean)
Write thread log files

Default: 0

### threads (integer)
global default thread count

Synonym: gthreads

Default: 0

### threadzerosync (integer)
Which CPLEX threads to use for synchronization

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Synchronize with multiple CPLEX threads</td>
</tr>
<tr>
<td>1</td>
<td>Only synchronize with CPLEX thread 0</td>
</tr>
</tbody>
</table>

### tilim (real)
overrides the GAMS ResLim option

Default: 1.0e+75

### trelim (real)
maximum space in memory for tree

Default: 1.0e+75

### tryint (real)
GMO tryint

Default: 0.0

### tuningdettlim (real)
tuning deterministic time limit per model or suite

Default: 1.0e+75

### tuningdisplay (integer)
level of information reported by the tuning tool

Default: 1

### tuningmeasure (integer)
measure for evaluating progress for a suite of models

Default: 1

### tuningrepeat (integer)
number of times tuning is to be repeated on perturbed versions

Default: 1

### tuningtilim (real)
tuning time limit per model or suite

Default: 1.0e+75

### usebasis (boolean)
GMO usebasis

Default: 0

### variableclean (integer)
Clean variable values from sub-models

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No cleaning</td>
</tr>
<tr>
<td>1</td>
<td>Quick cleaning and allow feasible uncleaned solutions if unable to clean</td>
</tr>
<tr>
<td>2</td>
<td>Quick cleaning and disallow uncleaned solutions</td>
</tr>
<tr>
<td>3</td>
<td>Thorough cleaning</td>
</tr>
</tbody>
</table>

varsel (integer): variable selection strategy at each node

Default: 0

warninglimit (integer): determines how many times warnings of a specific type (datacheck=2) will be displayed

Default: 10

workdir (string): directory for working files

Default: .

workeralgorithm (integer): set method for optimizing benders subproblems

Default: 0

workmem (real): memory available for working storage

Default: 2048.0

writeannotation (string): produce a Cplex annotation file

writebas (string): produce a Cplex basis file

writeflt (string): produce a Cplex solution pool filter file

writelp (string): produce a Cplex LP file

writemps (string): produce a Cplex MPS file

writemst (string): produce a Cplex mst file

writeord (string): produce a Cplex ord file

writeparam (string): produce a Cplex parameter file with all active options

writepre (string): produce a Cplex LP/MPS/SAV file of the presolved problem

writeprob (string): produce a Cplex problem file and inferrrs the type from the extension

writesav (string): produce a Cplex binary problem file

zerohalfcuts (integer): zero-half cuts

Default: 0

zerotol (real): Zero tolerance for variable values

Default: 1e-9
5.31 OSICPLEX, OSIGUROBI, OSIMOSEK, OSIXPRESS

The "bare bone" solver links GAMS/OSICPLEX, GAMS/OSIGUROBI, GAMS/OSIMOSEK, and GAMS/OSIXPRESS allow users to solve their GAMS models with a standalone license of CPLEX, GUROBI, MOSEK, or XPRESS. The links use the COIN-OR Open Solver Interface (OSI) to communicate with these solvers. The OSICPLEX link has been written primarily by Tobias Achterberg, the OSIGUROBI link has been written primarily by Stefan Vigerske, the OSIMOSEK link has been written primarily by Bo Jensen, and the OSIXPRESS link has been written primarily by John Doe. Matthew Saltzman is the COIN-OR project leader for OSI.

For more information we refer to the OSI web site.

The OSI links support linear equations and continuous, binary, and integer variables. The following list provides a brief overview on features available with GAMS/Solver and GAMS/Solver-Link licenses that are not available with the bare bone OSI solver links:

- Nonlinear model types such as QCP, MIQCP, NLP and MINLP
- Variable types SemiInt, SemiCont, SOS1 and SOS2
- certain GAMS/Solver options have no OSI counterpart (see e.g. the Comparison of GAMS/CPLEX and GAMS/OSICPLEX options)

5.31.1 Usage

The following statement can be used inside your GAMS program to specify using OSIGUROBI

Option LP = OSIGUROBI; { or MIP or RMIP }

Similar statements apply to OSICPLEX, OSIMOSEK, and OSIXPRESS.

The above statement should appear before the Solve statement.

The links support the general GAMS options reslim (time limit), iterlim (iteration limit), nodlim (node limit), optca (absolute gap tolerance), optcr (relative gap tolerance), cheat (cutoff decrement, only Xpress), and threads. An option file in the format required by the solver can be provided via the GAMS optfile option. See Section Option files for details.

If a MIP is solved via one of the OSI links, only primal solution values are reported by default. To receive also the dual values for the LP that is obtained from the MIP by fixing all discrete variables, the GAMS option integer1 must be set to a nonzero value. Note that this may lead to solving another LP after the MIP solve has finished.

Setting the GAMS option integer2 to a nonzero value makes variable and equation names available to the solver. This option may be useful for debugging purposes.

Setting the GAMS option integer3 to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process (integer3=1 writes an MPS file, integer3=2 writes an LP files, integer3=4 writes a native MPS file; sum these values to write several files). The name of the MPS file is chosen to be the name of the GAMS model file with the extension .gms replaced by .mps. This option may be useful for debugging purposes.

For OSICPLEX, OSIGUROBI, and OSIXPRESS, setting the GAMS option integer4 to a nonzero value leads to passing the variable level values (.l suffix) to the MIP solver as initial solution. This is analog to mipstart option of the full CPLEX and GUROBI links and the loadmipsol option of the full XPRESS link.
5.31.1 Option files

5.31.1.1 OSICPLEX Options  In an OSICPLEX option file, each line lists one option setting, where the option name and value are separated by space.

Example:

```
CPX_PARAM_MIPEMPHASIS 2
CPX_PARAM_HEURFREQ 42
CPX_PARAM_MIPDISPLAY 4
```

Note that GAMS/OSICPLEX will not process a GAMS/CPLEX option file. Instead, native CPLEX options (CPX...) have to be provided through an OSICPLEX option file (e.g. osicplex.opt) that follows the very strict syntax of a CPLEX parameter file. There is a trick to convert GAMS/CPLEX options into native CPLEX options: Create a regular GAMS/CPLEX option file cplex.opt with the options you would like to set for GAMS/OSICPLEX and add the option writeparam osicplex.opt. For example

```
* Cplex.opt
varsele 3 ! set variable selection to strong branching
writeparam osicplex.opt
```

Run this on any demo sized MIP problem (e.g. [MAGIC] from the GAMS Model Library) using GAMS/CPLEX. After the run you will find a file that looks as follows and can be supplied to GAMS/OSICPLEX to set the variable selection to strong branching (native CPLEX option CPXPARAM_MIP_Strategy_VariableSelect 3):

```
CPLEX Parameter File Version 12.10.0.0
CPXPARAM_Advance 0
CPXPARAM_Threads 1
CPXPARAM_Parallel 1
CPXPARAM_MIP_Display 4
CPXPARAM_MIP_Strategy_VariableSelect 3
CPXPARAM_MIP_Tolerances_AbsMIPGap 0
CPXPARAM_MIP_Tolerances_MIPGap 0
CPXPARAM_WorkDir "C:\[...]\Documents\GAMS\Studio\workspace\225a"
```

This includes some CPLEX options that have been set by the GAMS/CPLEX link and you might want to remove from this file.

Note that several GAMS/CPLEX options have no GAMS/OSICPLEX counterpart:

- **solnpool** (Solution Pool)
- **feasopt**
- **iis**
- **objrng, rhsrng** (Sensitivity Analysis)
- **indic** (Indicator constraints)
- **tuning** (Automatic Option Tuning)
- BCH Facility Options
- **quality** (Statistics of Solution)
- **writeXXX** (create some Cplex specific output files, e.g. XXX=sav)
- **tryInt**
- **interactive**
- **fixoptfile**
- **miptrace**
5.31 OSICPLEX, OSIGUROBI, OSIMOSEK, OSIXPRESS

5.31.1.1.2 OSIGUROBI Options  In an OSIGUROBI option file, each line lists one option setting, where the option name and value are separated by space.

Example:

Cuts 2
Heuristics 0.1

5.31.1.1.3 OSIMOSEK Options  An OSIMOSEK option file begins with the line BEGIN MOSEK and terminates with END MOSEK. Comments are introduced with an '%', empty lines are ignored. Each other line starts with a MOSEK parameter value, followed by space, and a value for that parameter.

Example:

BEGIN MOSEK
% disable probing and solve the root node by the interior point solver
MSK_IPAR_MIO_PRESOLVE_PROBING MSK_OFF
MSK_IPAR_MIO_ROOT_OPTIMIZER MSK_OPTIMIZER_INTPNT
END MOSEK

5.31.1.1.4 OSIXPRESS Options  In an OSIXPRESS option file, each line lists one option setting, where the option name and value are separated by an equal sign.

Example:

MIPLOG = 3
HEURFREQ = 2

5.31.1.2 Setup

5.31.1.2.1 OSICPLEX Setup  The setup for GAMS/OSICPLEX is identical to the GAMS/CPLEX-Link setup but to use the bare-bone interface GAMS/OSICPLEX, you do not need a GAMS/CPLEX-Link license but instead you have to request a free of charge GAMS/OSICPLEX license from sales@gams.com.

5.31.1.2.2 OSIGUROBI Setup  The setup for GAMS/OSIGUROBI is identical to the GAMS/Gurobi-Link setup but to use the bare-bone interface GAMS/OSIGUROBI, you do not need a GAMS/GUROBI-Link license.

5.31.1.2.3 OSIMOSEK Setup  The setup for GAMS/OSIMOSEK is identical to the GAMS/MOSEK-Link setup but to use the bare-bone interface GAMS/OSIMOSEK, you do not need a GAMS/MOSEK-Link license.

5.31.1.2.4 OSIXPRESS Setup  The setup for GAMS/OSIXPRESS is identical to the GAMS/XPRESS-Link setup but to use the bare-bone interface GAMS/OSIXPRESS, you do not need a GAMS/XPRESS-Link license.
5.32 PATHNLP

5.32.1 Introduction

This document describes the GAMS/PATHNLP solver for non-linear programs and the options unique to this solver.

PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems. The solution to the original NLP is extracted from the KKT solution and returned to GAMS. All of this takes place automatically - no special syntax or user reformulation is required.

Typically, PATHNLP works very well for convex models. It also has a comparative advantage on models whose solution via reduced gradient methods results in a large number of superbasic variables, since the PATH solver won't construct a dense reduced Hessian in the space of the superbasic variables as reduced gradient solvers do. For nonconvex models, however, PATHNLP is not as robust as the reduced gradient methods.

The theory relating NLP to their KKT systems is well-known: assuming differentiability without convexity, and assuming a constraint qualification holds, then a solution to the NLP must also be a solution to the KKT system. If we also assume convexity, then a solution to the KKT system is also a solution to the NLP - no further constraint qualification is required.

In case PATH fails to find a solution to the KKT system for the NLP, a phase I / phase II method is used in which the phase I objective is simply the feasibility error and the original objective is ignored. If a feasible point is found in phase I then phase II, an attempt to solve the KKT system for the NLP using the current feasible point, is entered.

PATHNLP is installed automatically with your GAMS system. With a demo or community license, it will solve small models only). If your GAMS license includes PATH, this size restriction is removed.

5.32.2 Usage

If you have installed the system and configured PATHNLP as the default NLP solver, all NLP models without a specific solver option will be solved with PATHNLP. If you installed another solver as the default, you can explicitly request that a particular model be solved using PATHNLP by inserting the statement

\texttt{option NLP = pathnlp;}

somewhere before the solve statement. Similar comments hold for the other model types (LP, RMINLP, QCP, etc.) PATHNLP can handle.

The standard GAMS model options iterlim and reslim can be used to control PATHNLP. A description of these options can be found in the GAMS Options section of the chapter on basic solver usage. In general this is enough to use PATHNLP effectively. In some cases, however, you may want to use some of the PATH or PATHNLP options to gain further performance improvements or for other reasons. The rules for using an option file are described in the chapter on basic solver usage. The options used to control PATH can also be used to control PATHNLP. There are also some options unique to PATHNLP.

5.32.3 Options

The tables that follow describe the options unique to PATHNLP as well as the options shared with the PATH solver for MCP models.

5.32.3.1 General options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>chen_lambda</td>
<td>lambda parameter for Chen-Chen-Kanzow residual</td>
<td>0.8</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>stopping criterion</td>
<td>1e-6</td>
</tr>
<tr>
<td>crash_iteration_limit</td>
<td>maximum iterations allowed in crash</td>
<td>50</td>
</tr>
<tr>
<td>crash_merit_function</td>
<td>merit function used in crash method</td>
<td>fischer</td>
</tr>
<tr>
<td></td>
<td>normal: Use the normal map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fischer: Use the Fischer function</td>
<td></td>
</tr>
<tr>
<td>crash_method</td>
<td>pnewton or none</td>
<td>pnewton</td>
</tr>
<tr>
<td></td>
<td>pnewton: Use projected Newton method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>crash_minimum_dimension</td>
<td>minimum problem dimension to perform crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_nbchange_limit</td>
<td>number of changes to the basis allowed</td>
<td>1</td>
</tr>
<tr>
<td>crash_perturb</td>
<td>perturb the problem using pnewton crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_searchtype</td>
<td>search type to use in the crash method</td>
<td>line</td>
</tr>
<tr>
<td></td>
<td>line: Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc: Use an arcsearch</td>
<td></td>
</tr>
<tr>
<td>cumulative_iteration_limit</td>
<td>maximum minor iterations allowed</td>
<td>10000</td>
</tr>
<tr>
<td>gradient_searchtype</td>
<td>search type to use on a gradient step</td>
<td>arc</td>
</tr>
<tr>
<td></td>
<td>line: Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc: Use an arcsearch</td>
<td></td>
</tr>
<tr>
<td>gradient_step_limit</td>
<td>gradient steps allowed before restarting</td>
<td>5</td>
</tr>
<tr>
<td>interrupt_limit</td>
<td>ctrl-C’s required before killing job</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>major_iteration_limit</td>
<td>maximum major iterations allowed</td>
<td>500</td>
</tr>
<tr>
<td>merit_function</td>
<td>merit function to use (normal or fischer)</td>
<td>fischer</td>
</tr>
<tr>
<td></td>
<td>normal: Use the normal map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fischer: Use the Fischer function</td>
<td></td>
</tr>
<tr>
<td>minor_iteration_limit</td>
<td>minor iterations allowed in each major iteration</td>
<td>10000</td>
</tr>
<tr>
<td>nms</td>
<td>allow line searching, watch-dogging, and nonmonotone descent</td>
<td>1</td>
</tr>
<tr>
<td>nms_initial_reference_factor</td>
<td>controls size of initial reference value</td>
<td>20</td>
</tr>
<tr>
<td>nms_maximum_watchdogs</td>
<td>maximum number of watchdog steps allowed</td>
<td>5</td>
</tr>
<tr>
<td>nms_memory_size</td>
<td>number of reference values kept</td>
<td>10</td>
</tr>
<tr>
<td>nms_mstep_frequency</td>
<td>frequency at which m-steps are performed</td>
<td>10</td>
</tr>
<tr>
<td>nms_searchtype</td>
<td>search type to use</td>
<td>line</td>
</tr>
<tr>
<td></td>
<td>line: Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc: Use an arcsearch</td>
<td></td>
</tr>
<tr>
<td>option_file</td>
<td>option file name for PATHLIB to read</td>
<td></td>
</tr>
<tr>
<td>preprocess</td>
<td>turns preprocessing on/off</td>
<td>1</td>
</tr>
<tr>
<td>proximal_perturbation</td>
<td>initial perturbation</td>
<td>0</td>
</tr>
<tr>
<td>time_limit</td>
<td>number of seconds algorithm is allowed to run</td>
<td></td>
</tr>
<tr>
<td>lemke_rank_deficiency_iterations</td>
<td>number of attempts made to fix rank-deficient basis during Lemke start</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>lemke_start</td>
<td>frequency of lemke starts &lt;br/&gt;always: Use a Lemke start for each LCP subproblem &lt;br/&gt;automatic: Determined by algorithm &lt;br/&gt;first: Use a Lemke start for the first LCP subproblem</td>
<td>automatic</td>
</tr>
<tr>
<td>lemke_start_type</td>
<td>type of lemke start &lt;br/&gt;advanced: Start Lemke method using an advanced basis &lt;br/&gt;slack: Start Lemke method using and all-slay basis</td>
<td>slack</td>
</tr>
</tbody>
</table>

5.32.3.2 NLP-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_reform</td>
<td>substitute out objective var and equ when possible &lt;br/&gt;Many models have an objective variable and equation that can be substituted out of the model, e.g. f(x) =E= z; If this option is true, PATHNLP will substitute out the objective variable and equation where possible.</td>
<td>1</td>
</tr>
<tr>
<td>gmo_hess_factor</td>
<td>maximum multiples of Jacobian size to allow Hessian storage: 0=no limit</td>
<td>0</td>
</tr>
<tr>
<td>nlp_lambda</td>
<td>linesearch factor when using the NLP objective &lt;br/&gt;If nlp_objective is true and nlp_lambda is positive, the PATH linesearch will be altered to take the objective function into account.</td>
<td>0</td>
</tr>
<tr>
<td>nlp_objective</td>
<td>treat NLP objective differently in PATH linesearch</td>
<td>0</td>
</tr>
<tr>
<td>output_memory</td>
<td>output breakdown of where memory is used</td>
<td>0</td>
</tr>
<tr>
<td>skip_kkt</td>
<td>go right to Phase I / Phase II method &lt;br/&gt;If true, PATHNLP will skip the initial attempt to solve the KKT system for the NLP and go directly into a Phase I / Phase II method that first attempts to get feasible and then attempts to solve the KKT system starting from the feasible point found in Phase I.</td>
<td>0</td>
</tr>
</tbody>
</table>

5.32.3.3 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_crash_iterations</td>
<td>output information on crash iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_crash_iterations_frequency</td>
<td>frequency at which crash iteration log is printed &lt;br/&gt;Range: {1, ..., ∞}</td>
<td>1</td>
</tr>
<tr>
<td>output_errors</td>
<td>output error messages</td>
<td>1</td>
</tr>
<tr>
<td>output_final_degeneracy_statistics</td>
<td>print information regarding degeneracy at the solution</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point</td>
<td>output final point returned from PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point_statistics</td>
<td>output information about the point, function, and Jacobian at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the final point</td>
<td>0</td>
</tr>
<tr>
<td>output_final_statistics</td>
<td>output evaluation of available merit functions at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_summary</td>
<td>output summary information</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>output_initial_point</td>
<td>output initial point given to PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_initial_point_statistics</td>
<td>output information about the point, function, and Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_statistics</td>
<td>output evaluation of available merit functions at the initial point</td>
<td>0</td>
</tr>
<tr>
<td>output_linear_model</td>
<td>output linear model at each major iteration</td>
<td>0</td>
</tr>
<tr>
<td>output_major_iterations</td>
<td>output information on major iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_major_iterations_frequency</td>
<td>frequency at which major iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td>output_maximum_zero_listing</td>
<td>limits zero columns reported to listing file</td>
<td>1000</td>
</tr>
<tr>
<td>output_maximum_zero_log</td>
<td>limits zero columns reported to log file</td>
<td>10</td>
</tr>
<tr>
<td>output_minor_iterations</td>
<td>output information on minor iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_minor_iterations_frequency</td>
<td>frequency at which minor iteration log is printed</td>
<td>500</td>
</tr>
<tr>
<td>output_options</td>
<td>output all options and their values</td>
<td>0</td>
</tr>
<tr>
<td>output</td>
<td>no turns all output off</td>
<td>1</td>
</tr>
<tr>
<td>output_preprocess_level</td>
<td>control output of preprocessing information</td>
<td>1</td>
</tr>
<tr>
<td>output_restart_log</td>
<td>output options during restarts</td>
<td>1</td>
</tr>
<tr>
<td>output_time</td>
<td>output breakdown of where time is spent</td>
<td>0</td>
</tr>
<tr>
<td>output_warnings</td>
<td>output warning messages</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.33 PATH

**Author**

Michael C. Ferris  
Todd S. Munson

### 5.33.1 Complementarity

A fundamental problem of mathematics is to find a solution to a square system of nonlinear equations. Two generalizations of nonlinear equations have been developed, a constrained nonlinear system which incorporates bounds on the variables, and the complementarity problem (MCP). This document is primarily concerned with the complementarity problem.

The PATH solver for MCP models is a Newton-based solver that combines a number of the most effective variations, extensions, and enhancements of this powerful technique. See PATH vs MILES for a comparison with MILES. Algorithmic details can also be found in papers and technical reports by Dirkse, Ferris, and Munson on [Ferris' Home Page](#).
The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. In its simplest form, the combinatorial problem is to choose from 2n inequalities a subset of n that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and economics [60] where we might want to compute Wardropian and Walrasian equilibria, and optimization where we can model the first order optimality conditions for nonlinear programs [102] [114]. Other examples, such as bimatrix games [117] and options pricing [97], abound.

Our development of complementarity is done by example. We begin by looking at the optimality conditions for a transportation problem and some extensions leading to the nonlinear complementarity problem. We then discuss a Walrasian equilibrium model and use it to motivate the more general mixed complementarity problem. We conclude this chapter with information on solving the models using the PATH solver and interpreting the results.

### 5.33.1.1 Transportation Problem

The transportation model is a linear program where demand for a single commodity must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs, where $(i, j) \in \mathcal{A}$ means that there is a route from supplier $i$ to demand center $j$. The problem variables are the quantities $x_{i,j}$ shipped over each arc $(i, j) \in \mathcal{A}$. The linear program can be written mathematically as

$$
\begin{align*}
&\min_{x \geq 0} \sum_{(i,j) \in \mathcal{A}} c_{i,j} x_{i,j} \\
&\text{subject to} \quad \sum_{j : (i,j) \in \mathcal{A}} x_{i,j} \leq s_i, \forall i \\
&\quad \sum_{i : (i,j) \in \mathcal{A}} x_{i,j} \geq d_j, \forall j.
\end{align*}
$$

(1)

where $c_{i,j}$ is the unit shipment cost on the arc $(i, j)$, $s_i$ is the available supply at $i$, and $d_j$ is the demand at $j$.

The derivation of the optimality conditions for this linear program begins by associating with each constraint a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. We label the prices on the supply constraint $p^s$ and those on the demand constraint $p^d$. Intuitively, for each supply node $i$

$$0 \leq p^s_i, \quad s_i \geq \sum_{j : (i,j) \in \mathcal{A}} x_{i,j},$$

Consider the case when $s_i > \sum_{j : (i,j) \in \mathcal{A}} x_{i,j}$, that is there is excess supply at $i$. Then, in a competitive marketplace, no rational person is willing to pay for more supply at node $i$; it is already over-supplied. Therefore, $p^s_i = 0$. Alternatively, when $s_i = \sum_{j : (i,j) \in \mathcal{A}} x_{i,j}$, that is node $i$ clears, we might be willing to pay for additional supply of the good. Therefore, $p^s_i \geq 0$. We write these two conditions succinctly as:

$$0 \leq p^s_i \quad \perp \quad s_i \geq \sum_{j : (i,j) \in \mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p^s_i$, the first case, or $s_i = \sum_{j : (i,j) \in \mathcal{A}} x_{i,j}$, the second case.

Similarly, at each node $j$, the demand must be satisfied in any feasible solution, that is

$$\sum_{i : (i,j) \in \mathcal{A}} x_{i,j} \geq d_j.$$

Furthermore, the model assumes all prices are nonnegative, $0 \leq p^d_j$. If there is too much of the commodity supplied, $\sum_{i : (i,j) \in \mathcal{A}} x_{i,j} > d_j$, then, in a competitive marketplace, the price $p^d_j$ will be driven down to 0. Summing these relationships gives the following complementarity condition:

$$0 \leq p^d_j \quad \perp \quad \sum_{i : (i,j) \in \mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$

The supply price at $i$ plus the transportation cost $c_{i,j}$ from $i$ to $j$ must exceed the market price at $j$. That is, $p^s_i + c_{i,j} \geq p^d_j$. Otherwise, in a competitive marketplace, another producer will replicate supplier.
increasing the supply of the good in question which drives down the market price. This chain would repeat until the inequality is satisfied. Furthermore, if the cost of delivery strictly exceeds the market price, that is 
\[ p^s_i + c_{i,j} > p^d_j, \] 
then nothing is shipped from \( i \) to \( j \) because doing so would incur a loss and \( x_{i,j} = 0 \). Therefore,

\[ 0 \leq x_{i,j} \perp p^s_i + c_{i,j} \geq p^d_j, \quad \forall (i,j) \in A. \]

We combine the three conditions into a single problem,

\[ \begin{align*}
0 & \leq p^s_i \perp s_i \geq \sum_{j:(i,j) \in A} x_{i,j}, \quad \forall i \\
0 & \leq p^d_j \perp \sum_{i:(i,j) \in A} x_{i,j} \geq d_j, \quad \forall j \\
0 & \leq x_{i,j} \perp p^s_i + c_{i,j} \geq p^d_j, \quad \forall (i,j) \in A.
\end{align*} \tag{2} \]

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions \[37\] of the linear program (1). For linear programs the complementary slackness conditions are both necessary and sufficient for \( x \) to be an optimal solution of the problem (1). Furthermore, the conditions (2) are also the necessary and sufficient optimality conditions for a related problem in the variables \((p^s, p^d)\)

\[
\begin{array}{ll}
\max_{p^s, p^d \geq 0} & \sum_j d_j p^d_j - \sum_i s_i p^s_i \\
\text{subject to} & c_{i,j} \geq p^d_j - p^s_i, \quad \forall (i,j) \in A
\end{array}
\]
termed the dual linear program (hence the nomenclature "dual variables").

Looking at (2) a bit more closely we can gain further insight into complementarity problems. A solution of (2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However, the key to the modeling power of complementarity is that it chooses which of the inequalities in (2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active. A regime shift could, for example, be a back stop technology like windmills that become profitable if a \( CO_2 \) tax is increased.

5.33.1.1.1 GAMS Code  The GAMS code for the complementarity version of the transportation problem is given in Figure 1; the actual data for the model is assumed to be given in the file transmcp.dat. Note that the model written corresponds very closely to (2). In GAMS, the \( \perp \) sign is replaced in the model statement with a ".". It is precisely at this point that the pairing of variables and equations shown in (2) occurs in the GAMS code. For example, the function defined by \emph{rational} is complementary to the variable \( x \). To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable \( z(i) \)):

\[
z \text{.lo(i)} = 0;
\]
or alternatively

\[
\text{positive variable } z;
\]

Further information on the GAMS syntax can be found in [155]. Note that \emph{GAMS} requires the modeler to write \( F(z) \) \( \geq 0 \) whenever the complementary variable is lower bounded, and does not allow the alternative form \( 0 = l = F(z) \).

\textbf{Figure 1:} A simple MCP model in GAMS, \texttt{transmcp.gms}
sets  i  canning plants,
    j  markets ;

parameter
    s(i)  capacity of plant i in cases,
    d(j)  demand at market j in cases,
    c(i,j)  transport cost in thousands of dollars per case ;

$include transmcp.dat

positive variables
    x(i,j)  shipment quantities in cases
    p_demand(j)  price at market j
    p_supply(i)  price at plant i;

equations
    supply(i)  observe supply limit at plant i
    demand(j)  satisfy demand at market j
    rational(i,j) ;

supply(i) ..  s(i) =g= sum(j, x(i,j)) ;

demand(j) ..  sum(i, x(i,j)) =g= d(j) ;

rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;

model transport / rational.x, demand.p_demand, supply.p_supply /;
solve transport using mcp;

5.33.1.1.2 Extension: Model Generalization

While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods [205]), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1 is independent of the prices \( p \). Since the prices \( p \) are variables in the complementarity problem (2), we can easily replace the constant demand \( d \) with a function \( d(p) \) in the complementarity setting. Clearly, any algebraic function of \( p \) that can be expressed in GAMS can now be added to the model given in Figure 1. For example, a linear demand function could be expressed using

\[
\sum_{i:(i,j)\in A} x_{i,j} \geq d_j(1 - p^d_j), \forall j.
\]

Note that the demand is rather strange if \( p^d_j \) exceeds 1. Other more reasonable examples for \( d(p) \) are easily derived from Cobb-Douglas or CES utilities. For those examples, the resulting complementarity problem becomes nonlinear in the variables \( p \). Details of complementarity for more general transportation models can be found in [48], [62].

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (2) is replaced by

\[
p^t_i(1 + t_i) + c_{i,j} \geq p^d_j, \forall (i,j) \in A.
\]

The taxes can be made endogenous to the model, details are found in [155].

The key point is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In many cases, there is no optimization problem corresponding to the complementarity conditions.
5.33.1.1.3 Nonlinear Complementarity Problem  We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast as nonlinear complementarity problems (NCPs) defined as follows:

\[(\text{NCP})\] Given a function \(F : \mathbb{R}^n \to \mathbb{R}^n\), find \(z \in \mathbb{R}^n\) such that

\[0 \leq z \perp F(z) \geq 0.\]

Recall that the \(\perp\) sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, \(z_i F_i(z) = 0\). We frequently refer to this property as \(z_i\) is complementary to \(F_i\). A special case of the NCP that has received much attention is when \(F\) is a linear function, the linear complementarity problem [40].

5.33.1.2 Walrasian Equilibrium

A Walrasian equilibrium can also be formulated as a complementarity problem (see [132]). In this case, we want to find a price \(p \in \mathbb{R}^m\) and an activity level \(y \in \mathbb{R}^n\) such that

\[
0 \leq y \perp L(p) := -A^T p \geq 0 \\
0 \leq p \perp S(p, y) := b + Ay - d(p) \geq 0
\]

(3)

where \(S(p, y)\) represents the excess supply function and \(L(p)\) represents the loss function. Complementarity allows us to choose the activities \(y_j\) to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras’ law which states that supply equals demand if we assume all prices \(p\) will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter.

5.33.1.2.1 GAMS Code  A GAMS implementation of (3) is given in Figure 2. Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [95].

Figure 2: Walrasian equilibrium as an NCP, walras1.gms

$include walras.dat

positive variables p(i), y(j);
 equations S(i, L(j);
 S(i).. b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, g(k)*p(k)) / p(i) / =g= 0;
 L(j).. -sum(i, p(i)*A(i,j)) =g= 0;
 model walras / S.p, L.y /
 solve walras using mcp;
5.33.1.2.2 Extension: Intermediate Variables In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function \( d(p) \) in the Walrasian equilibrium (3). The syntax for carrying this out is shown in Figure 3 where we use the variables \( d \) to store the demand function referred to in the excess supply equation. The model \texttt{walras} now contains a mixture of equations and complementarity constraints. Since constructs of this type are prevalent in many practical models, the GAMS syntax allows such formulations.

**Figure 3:** Walrasian equilibrium as an MCP, \texttt{walras2.gms}

```gams
$include walras.dat

positive variables p(i), y(j);
variables d(i);
equations S(i), L(j), demand(i);

demand(i).. 
    d(i) =e= c(i)*sum(k, g(k)*p(k)) / p(i) ;
S(i).. b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;
L(j).. -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;
```

Note that positive variables are paired with inequalities, while free variables are paired with equations. A crucial point misunderstood by many modelers (new and experienced alike) is that the bounds on the variable determine the relationships satisfied by the function \( F \). Thus in Figure 3, \( d \) is a free variable and therefore its paired equation \texttt{demand} is an equality. Similarly, since \( p \) is nonnegative, its paired relationship \texttt{S} is a (greater-than) inequality. See the MCP definition below for details.

A simplification is allowed to the model statement in Figure 3. It is not required to match free variables explicitly to equations; we only require that there are the same number of free columns (i.e. single variables) as unmatched rows (i.e. single equations). Thus, in the example of Figure 3, the model statement could be replaced by

```gams
model walras / demand, S.p, L.y /;
```

This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged. GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equality rows equals the number of remaining free columns. However, if an explicit match is given, the PATH solver can frequently exploit the information for better solution. Note that all variables that are not free and all inequalities must be explicitly matched.

5.33.1.2.3 Mixed Complementarity Problem A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds \( \ell \), the upper bounds \( u \) and the function \( F \).

\[ \text{(MCP)} \text{ Given lower bounds } \ell \in \{ \mathbb{R} \cup \{-\infty\}\}^n, \text{ upper bounds } u \in \{ \mathbb{R} \cup \{\infty\}\}^n \text{ and a function } F : \mathbb{R}^n \to \mathbb{R}^n, \text{ find } z \in \mathbb{R}^n \text{ such that precisely one of the following holds for each } i \in \{1, \ldots, n\}: \]

\[
F_i(z) = 0 \quad \text{and} \quad \ell_i \leq z_i \leq u_i \\
F_i(z) > 0 \quad \text{and} \quad z_i = \ell_i \\
F_i(z) < 0 \quad \text{and} \quad z_i = u_i.
\]
These relationships define a general MCP (sometimes termed a rectangular variational inequality [94]). We will write these conditions compactly as

\[ \ell \leq x \leq u \perp F(x). \]

As the MCP model type as defined above is the one used in GAMS, there are some consequences of the definition worth emphasizing here. Firstly, the bounds on the variable determine the relationships satisfied by the function \( F \): the constraint type (e.g. \( =G=, =L= \)) chosen plays no role in defining the problem. This being the case, it is acceptable (and perhaps advisable!) to use an \( =N= \) in the definition of all equations used in complementarity models to highlight the dependence on the variable bounds in defining the MCP. This is not required, however. The constraint types \( =G=, =L= \) and \( =E= \) are also allowed. If the variable bounds are inconsistent with the constraint type chosen, GAMS will either reject the model (e.g. an \( =L= \) row matched with a lower-bounded variable) or silently and temporarily ignore the issue (e.g. an \( =E= \) row matched with a lower-bounded variable). In the latter case, if equality does not hold at solution (allowable if the variable is at lower bound) the is marked in the listing file with a redef.

Secondly, note the contrast between complementarity conditions like \( g(z) \leq 0, z \geq 0 \) and a complementarity problem like \( F(x) \perp 0 \leq x \leq \infty \). The former requires a trivial transformation in order to fit into the MCP framework: \(-g(z) \perp 0 \leq z \leq \infty \). This simple issue is a frequent source of problems and worth looking out for.

The nonlinear complementarity problem of Nonlinear Complementarity Problem is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

\[ z.\text{lo}(I) = 0; \]

or declare

positive variable z;

Another special case is a square system of nonlinear equations

(NE) Given a function \( F : \mathbb{R}^n \rightarrow \mathbb{R}^n \) find \( z \in \mathbb{R}^n \) such that

\[ F(z) = 0. \]

In order to formulate this in the GAMS/MCP format we just declare

free variable z;

In both the above cases, we must not modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of \( F \). If a variable \( z_i \) is fixed, then \( F_i(z) \) is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when \( z_i = \ell_i = u_i \). Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time. As an example, in economics, fixing a level of production will remove the zero-profit condition for that activity.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

\[ \min f(x) \text{ subject to } \ell \leq x \leq u. \]
The first order optimality conditions for this problem class are precisely MCP(∇f(x), [ℓ, u]). We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then ∇f(x) = 0. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so ∇f(x) ≥ 0. Conversely, if x is at its upper bound, then the function must be increasing as x decreases, so that ∇f(x) ≤ 0. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in Figure 3 it may be necessary to have an upper bound on the activity level y. In this case, we simply add an upper bound to y in the model statement, and we can replace the loss equation with the following definition:

\[
y_{\text{up}}(j) = 10;
L(j) . \sum(i, p(i)A(i,j)) =e= 0 ;
\]

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less-than inequality or greater-than inequality, since this determination depends on the values of the solution variables. However, let us interpret the relationships that the above change generates. If y_j = 0, the loss function can be positive since we are not producing in the jth sector. If y_j is strictly between its bounds, then the loss function must be zero by the definition of complementarity; this is the competitive assumption. Finally, if y_j is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [45], [59], [60].

### 5.33.1.3 Solution

We will assume that a file named transmcp.gms has been created using the GAMS syntax which defines an MCP model transport as developed in Transportation Problem. The modeler has a choice of the complementarity solver to use. We are going to further assume that the modeler wants to use PATH.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the transmcp.gms file prior to the solve statement

   ```
   option mcp = path;
   PATH will then be used instead of the default solver provided.
   ```

2. Choose PATH as the default solver for MCP (via the IDE on Windows or elsewhere by rerunning gamsinst from the GAMS system directory).

To solve the problem, the modeler executes the command:

```
gams transmcp
```

where transmcp can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to List of Command Line Parameters for further details.

At this stage, control is handed over to the solver which creates a log providing information on what the solver is doing as time elapses. See Section PATH for details about the log file. After the solver terminates, a listing file is generated containing the solution to the problem. We now describe the listing file output specifically related to complementarity problems.
5.33.1.3.1 Listing File  The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output, an example of which is given in Figure 4. We use "..." to indicate where we have omitted continuing similar output.

**Figure 4:** Listing File for solving transmcp.gms

```plaintext
S O L V E       S U M M A R Y
MODEL TRANSPORT
TYPE MCP
SOLVER PATH FROM LINE 45

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 1 OPTIMAL

RESOURCE USAGE, LIMIT 0.057 1000.000
ITERATION COUNT, LIMIT 31 10000
EVALUATION ERRORS 0 0

Work space allocated -- 0.06 Mb

---- EQU RATIONAL

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle .new-york</td>
<td>-0.225</td>
<td>-0.225</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .chicago</td>
<td>-0.153</td>
<td>-0.153</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .topeka</td>
<td>-0.162</td>
<td>-0.126</td>
<td>+INF</td>
</tr>
</tbody>
</table>

...

---- VAR X shipment quantities in cases

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle .new-york</td>
<td>.</td>
<td>50.000</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .chicago</td>
<td>.</td>
<td>300.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

...

**** REPORT SUMMARY :
0 NONOPT
0 INFEASIBLE
0 UNBOUNDED
0 REDEFINED
0 ERRORS
```

After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. Table 1 and Table 2 display the relevant codes that are returned under different circumstances. A modeler can access these codes within the transmcp.gms file using transport.solvestat and transport.modelstat respectively.

**Table 1:** Solver Status Codes
After this, a listing of the time and iterations used is given, along with a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "****". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal. GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value of the equation (an evaluation of the left hand side of the constraint at the current point) should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle .chicago -0.153 -2.000 +INF 300.000 INFES
```

The marginal column in the equation contains the value of the variable that was matched with this equation.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of equ.l - equ.lower and equ.l - equ.upper, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. Figure 4 is nicely-solved case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

### 5.3.3.1.3.2 Redefined Equations

Unfortunately, this is not the end of the story. Some equations may have the following form:

```
LOWER   LEVEL   UPPER   MARGINAL
new-york 325.000 350.000 325.000 0.225 REDEF
```
This should be construed as a warning from GAMS, as opposed to an error. In principle, the \texttt{REDEF} should only occur if the paired variable to the constraint has a finite lower and/or upper bound and the variable is at one of those bounds. In this case, at the solution of the complementarity problem the sense of the constraint (e.g. $\neq E=\neq$) may not be satisfied. This occurs when GAMS constraints are used to define the function $F$. The bounds on each component of the function $F$ are derived from the bounds on the matching variable component, and these bounds may be inconsistent with those implied by the constraint type used. GAMS warns the user via the \texttt{REDEF} label when the solution found leads to a violated "constraint". To avoid \texttt{REDEF} warnings the $=N=$ can be used to define all complementarity functions.

\textit{Note that a \texttt{REDEF} is not an error, just a warning. The solver has solved the complementarity problem specified.} GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the variable bounds, not from the equation definition.

\subsection{Pitfalls}

As indicated above, the ordering of an equation is important in the specification of an MCP. Since the data of the MCP is the function $F$ and the bounds $\ell$ and $u$, it is important for the modeler to pass the solver the function $F$ and not $-F$.

For example, if we have the optimization problem,

$$
\min_{x \in [0, 2]} (x - 1)^2
$$

then the first order optimality conditions are

$$
0 \leq x \leq 2 \quad \perp \quad 2(x - 1)
$$

which has a unique solution, $x = 1$. Figure 5 provides correct GAMS code for this problem.

\textbf{Figure 5:} First order conditions as an MCP, \texttt{first.gms}

\begin{verbatim}
variables x;
equations d_f;
x.lo = 0;
x.up = 2;
d_f.. 2*(x - 1) =e= 0;
model first / d_f.x /;
solve first using mcp;
\end{verbatim}

However, if we accidentally write the valid equation

$$
d_f.. 0 =e= 2*(x - 1);
$$

the problem given to the solver is

$$
0 \leq x \leq 2 \quad \perp \quad -2(x - 1)
$$

which has three solutions, $x = 0$, $x = 1$, and $x = 2$. This problem is in fact the stationary conditions for the nonconvex quadratic problem,

$$
\max_{x \in [0, 2]} (x - 1)^2,
$$
not the problem we intended to solve.

Continuing with the example, when \( x \) is a free variable, we might conclude that the ordering of the equation is irrelevant because we always have the equation, \( 2(x - 1) = 0 \), which does not change under multiplication by \(-1\). In most cases, the ordering of equations (which are complementary to free variables) does not make a difference since the equation is internally "substituted out" of the model. In particular, for defining equations, such as that presented in Figure 3, the choice appears to be arbitrary.

However, in difficult (singular or near singular) cases, the substitution cannot be performed, and instead a perturbation is applied to \( F \), in the hope of "(strongly) convexifying" the problem. If the perturbation happens to be in the wrong direction because \( F \) was specified incorrectly, the perturbation actually makes the problem less convex, and hence less likely to solve. Note that determining which of the above orderings of the equations makes most sense is typically tricky. One rule of thumb is to check whether if you replace the "\( =e= \)” by "\( =g= \)” , and then increase "\( x \)” , is the inequality intuitively more likely to be satisfied. If so, you probably have it the right way round, if not, reorder.

Furthermore, underlying model convexity is important. For example, if we have the linear program

\[
\begin{align*}
\min_x & \quad c^T x \\
\text{subject to} & \quad Ax = b, \ x \geq 0 
\end{align*}
\]

we can write the first order optimality conditions as either

\[
0 \leq x \perp -A^T \mu + c \\
\mu \text{ free} \perp Ax - b
\]

or, equivalently,

\[
0 \leq x \perp -A^T \mu + c \\
\mu \text{ free} \perp b - Ax
\]

because we have an equation. The former is a linear complementarity problem with a positive semidefinite matrix, while the latter is almost certainly indefinite. Also, if we need to perturb the problem because of numerical problems, the former system will become positive definite, while the later becomes highly nonconvex and unlikely to solve. The rule of thumb here is to consider what happens when the dual multiplier \( \mu \) is made non-negative in the KKT system. In the former system, the constraint becomes \( Ax \geq b \), which is consistent with a minimization and a positive multiplier. In the latter system, we get \( Ax \leq b \), which is backwards.

Finally, users are strongly encouraged to match equations and free variables when the matching makes sense for their application. Structure and convexity can be destroyed if it is left to the solver to perform the matching. For example, in the above example, we could lose the positive semidefinite matrix with an arbitrary matching of the free variables.

### 5.33.2 PATH

Newton’s method, perhaps the most famous solution technique, has been extensively used in practice to solve square systems of nonlinear equations. The basic idea is to construct a local approximation of the nonlinear equations around a given point, \( x^k \), solve the approximation to find the Newton point, \( x^N \), update the iterate, \( x^{k+1} = x^N \), and repeat until we find a solution to the nonlinear system. This method works extremely well close to a solution, but can fail to make progress when started far from a solution. To guarantee progress is made, a line search between \( x^k \) and \( x^N \) is used to enforce sufficient decrease on an appropriately defined merit function. Typically, \( \frac{1}{2} \| F(x) \|^2 \) is used.

PATH uses a generalization of this method on a nonsmooth reformulation of the complementarity problem. To construct the Newton direction, we use the normal map [151] representation

\[
F(\pi(x)) + x - \pi(x)
\]
associated with the MCP, where \( \pi(x) \) represents the projection of \( x \) onto \( \ell, u \) in the Euclidean norm. We note that if \( x \) is a zero of the normal map, then \( \pi(x) \) solves the MCP. At each iteration, a linearization of the normal map, a linear complementarity problem, is solved using a pivotal code related to Lemke's method.

Versions of PATH prior to 4.x are based entirely on this formulation using the residual of the normal map

\[ \| F(\pi(x)) + x - \pi(x) \| \]

as a merit function. However, the residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. PATH 4.x considers an alternative nonsmooth system \[ \Phi(x) = 0, \] where \( \Phi(x) = \phi(x_i, F(x_i)) \) and \( \phi(a,b) := \sqrt{a^2 + b^2} - a - b \). The merit function, \( \| \Phi(x) \|^2 \), in this case is differentiable, and is used for globalization purposes. When the subproblem solver fails, a projected gradient direction for this merit function is searched. It is shown in [61] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP is provided under appropriate conditions.

The remainder of this chapter details the interpretation of output from PATH and ways to modify the behavior of the code. To this end, we will assume that the modeler has created a file named \texttt{transmcp.gms} which defines an MCP model \texttt{transport} as described in Section Transportation Problem and is using PATH 4.x to solve it. See Section Solution for information on changing the solver.

### 5.33.2.1 Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically produced. An example of the log for a particular run is given in Figure 6.

**Figure 6:** Log File from PATH for solving \texttt{transmcp.gms}

--- Starting compilation
--- trnsmcp.gms(46) 1 Mb
--- Starting execution
--- trnsmcp.gms(27) 1 Mb
--- Generating model transport
--- trnsmcp.gms(45) 1 Mb
--- 11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
  Work space allocated -- 0.06 Mb
Reading the matrix.
Reading the dictionary.
Path v4.3: GAMS Link ver037, SPARC/SOLARIS
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.3 (Sat Feb 26 09:38:08 2000)
Written by Todd Munson, Steven Dirkse, and Michael Ferris

**INITIAL POINT STATISTICS**
Maximum of X . . . . . . . . . . . . . . . -0.0000e+00 var: (x.seattle.new-york)
Maximum of F . . . . . . . . . . . . . . . 6.0000e+02 eqn: (supply.san-diego)
Maximum of Grad F . . . . . . . . . . . . 1.0000e+00 eqn: (demand.new-york)
  var: (x.seattle.new-york)

**INITIAL JACOBIAN NORM STATISTICS**
Maximum Row Norm . . . . . . . . . . 3.0000e+00 eqn: (supply.seattle)
Minimum Row Norm . . . . . . . . . . 2.0000e+00 eqn: (rational.seattle.new-york)
Maximum Column Norm . . . . . . . . 3.0000e+00 var: (p_supply.seattle)
Minimum Column Norm . . . . . . . . 2.0000e+00 var: (x.seattle.new-york)
Crash Log

<table>
<thead>
<tr>
<th>major</th>
<th>func</th>
<th>diff</th>
<th>size</th>
<th>residual</th>
<th>step</th>
<th>prox</th>
<th>(label)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.416e+03</td>
<td>0.0e+00</td>
<td></td>
<td></td>
<td>(demand.new-york)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1.0029e+03</td>
<td>1.0e+00</td>
<td>1.0e+01</td>
<td></td>
<td>(demand.new-york)</td>
</tr>
</tbody>
</table>

pn_search terminated: no basis change.

Figure 7: Log File from PATH for solving transmcp.gms (continued)

Major Iteration Log

<table>
<thead>
<tr>
<th>major</th>
<th>minor</th>
<th>func</th>
<th>grad</th>
<th>residual</th>
<th>step</th>
<th>type</th>
<th>prox</th>
<th>inorm</th>
<th>(label)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1.0029e+03</td>
<td>I</td>
<td>9.0e+00</td>
<td>6.2e+02</td>
<td></td>
<td>(demand.new-york)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>8.3096e+02</td>
<td>1.0e+00</td>
<td>SO</td>
<td>3.6e+00</td>
<td>4.5e+02</td>
<td>(demand.new-york)</td>
</tr>
</tbody>
</table>

... 15 2 17 17 1.3972e-09 1.0e+00 SO 4.8e-06 1.3e-09 (demand.chicago)

FINAL STATISTICS

Inf-Norm of Complementarity . . 1.4607e-08 eqn: (rational.seattle.chicago)
Inf-Norm of Normal Map . . . . 1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Minimum Map . . . . 1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Fischer Function . . 1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Grad Fischer Fcn . 1.3247e-09 eqn: (rational.seattle.chicago)

FINAL POINT STATISTICS

Maximum of X . . . . . . . . . . 3.0000e+02 var: (x.seattle.chicago)
Maximum of F . . . . . . . . . . 5.0000e+01 eqn: (supply.san-diego)
Maximum of Grad F . . . . . . . 1.0000e+00 eqn: (demand.new-york)
                          var: (x.seattle.new-york)

** EXIT - solution found. **

Major Iterations. . . . . 15
Minor Iterations. . . . 31
Restarts. . . . . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . . . 0.020000
Residual. . . . . . . . 1.397183e-09

--- Restarting execution

The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If PATH is directed to read an option file (see PATH Options below), then the following output is generated after the PATH banner.

Reading options file PATH.OPT
> output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts working on the problem.
5.33.2.1.1 Diagnostic Information  Some diagnostic information is initially generated by the solver at the starting point. Included is information about the initial point and function evaluation. The log file here tells the value of the largest element of the starting point and the variable where it occurs. Similarly, the maximum function value is displayed along with the row producing it. The maximum element in the gradient is also presented with the equation and variable where it is located.

The second block provides more information about the Jacobian at the starting point. This information can be used to help scale the model. See Section Advanced Topics for complete details.

5.33.2.1.2 Crash Log  The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in [47], and an example of the Crash Log can be seen in Figure 6. The first column of the crash log is just a label indicating the current iteration number, the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See Merit Functions for further information. The column "step" corresponds to the steplength taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section PATH Options gives mechanisms to affect the behavior of these steps.

5.33.2.1.3 Major Iteration Log  After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see Figure 7). The columns that have the same labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [117]. The number of pivots performed per major iteration is given in the "minor" column.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be taken, due to the use of nonmonotone searches [46], [56] which are used to improve robustness. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. Table 3 explains the return codes for the linear solver and Table 4 explains the meaning of each step type. The ideal output in this column is "SO"; "SD" and "SB" also indicate reasonable progress. Codes different from these are not catastrophic, but typically indicate the solver is having difficulties due to numerical issues or nonconvexities in the model.

Table 3: Linear Solver Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>A cycle was detected.</td>
</tr>
<tr>
<td>E</td>
<td>An error occurred in the linear solve.</td>
</tr>
<tr>
<td>I</td>
<td>The minor iteration limit was reached.</td>
</tr>
<tr>
<td>N</td>
<td>The basis became singular.</td>
</tr>
<tr>
<td>R</td>
<td>An unbounded ray was encountered.</td>
</tr>
<tr>
<td>S</td>
<td>The linear subproblem was solved.</td>
</tr>
<tr>
<td>T</td>
<td>Failed to remain within tolerance after factorization was performed.</td>
</tr>
</tbody>
</table>
### Table 4: Step Type Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.</td>
</tr>
<tr>
<td>D</td>
<td>The step was accepted because the Distance between the current iterate and the Newton point was small.</td>
</tr>
<tr>
<td>G</td>
<td>A gradient step was performed.</td>
</tr>
<tr>
<td>I</td>
<td>Initial information concerning the problem is displayed.</td>
</tr>
<tr>
<td>M</td>
<td>The step was accepted because the Merit function value is smaller than the nonmonotone reference value.</td>
</tr>
<tr>
<td>O</td>
<td>A step that satisfies both the distance and merit function tests.</td>
</tr>
<tr>
<td>R</td>
<td>A Restart was carried out.</td>
</tr>
<tr>
<td>W</td>
<td>A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the nonmonotone reference value (M, O, or B step), regenerated the Newton point, and performed a backtracking search.</td>
</tr>
</tbody>
</table>

#### 5.33.2.1.4 Minor Iteration Log

If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. A listing from transmcp model follows, where we have set the `output_minor_iteration_frequency` option to 1.

```plaintext
Minor Iteration Log
minor t z w v art ckpts enter leave
1 4.2538e-01 8 2 0 0 0 t[ 0] z[ 11]
2 9.0823e-01 8 2 0 0 0 w[ 10]
3 1.0000e+00 9 2 0 0 0 z[ 10] t[ 0]
```

t is a parameter that goes from zero to 1 for normal starts in the pivotal code. When the parameter reaches 1, we are at a solution to the subproblem. The t column gives the current value for this parameter. The next columns report the current number of problem variables z and slacks corresponding to variables at lower bound w and at upper bound v. Artificial variables are also noted in the minor log, see [57] for further details. Checkpoints are times where the basis matrix is refactorized. The number of checkpoints is indicated in the ckpts column. Finally, the minor iteration log displays the entering and leaving variables during the pivot sequence.

#### 5.33.2.1.5 Restart Log

The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When it is determined that no progress is being made, the problem is restarted from the initial point supplied in the GAMS file with a different set of options. These restarts give the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

The exact options set during the restart are given in the restart log, part of which is reproduced below.

```plaintext
Restart Log
proximal_perturbation 0
crash_method none
crash_perturb yes
nms_initial_reference_factor 2
proximal_perturbation 1.0000e-01
```

If a particular problem solves under a restart, a modeler can circumvent the wasted computation by setting the appropriate options as shown in the log. Note that sometimes an option is repeated in this log. In this case, it is the last option that is used.
5.33.2.1.6 Solution Log  A solution report is now given by the algorithm for the point returned. The first component is an evaluation of several different merit functions. Next, a display of some statistics concerning the final point is given. This report can be used detect problems with the model and solution as detailed in Section Advanced Topics.

At the end of the log file, summary information regarding the algorithm's performance is given. The string "** EXIT - solution found". is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what modelstat and solvestat will be returned to GAMS. After this, the "Restarting execution" flag indicates that GAMS has been restarted and is processing the results passed back by PATH.

5.33.2.2 Status File

If for some reason the PATH solver exits without writing a solution, or the sysout flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but provides more detailed information. The modeler is typically not interested in this output.

5.33.2.3 User Interrupts

A user interrupt can be effected by typing Ctrl-C. We only check for interrupts every major iteration. If a more immediate response is wanted, repeatedly typing Ctrl-C will eventually kill the job. The number needed is controlled by the interrupt_limit option. In this latter case, when a kill is performed, no solution is written and an execution error will be generated in GAMS.

5.33.2.4 Preprocessing

The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. A comprehensive preprocessor has been incorporated into PATHC as developed in [58].

The preprocessor reports its finding with some additional output to the log file. This output occurs before the initial point statistics. An example of the preprocessing on the forcebsm model is presented below.

    Zero:  0 Single:    112 Double:    0 Forced:    0
    Preprocessed size: 72

The preprocessor looks for special polyhedral structure and eliminates variables using this structure. These are indicated with the above line of text. Other special structure is also detected and reported.

On exit from the algorithm, we must generate a solution for the original problem. This is done during the postsolve. Following the postsolve, the residual using the original model is reported.

    Postsolved residual: 1.0518e-10

This number should be approximately the same as the final residual reported on the presolved model.
5.33.2.4.1 Constrained Nonlinear System  Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

\[ F(z) = 0, \ \ell \leq z \leq u, \]

where typically, the bounds on \( z \) are inactive at the solution. This is not an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function \( F \). If a finite bound is active at a solution, the corresponding component of \( F \) is only constrained to be nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do not satisfy \( F(z) = 0 \). Only if \( z^* \) is a solution of MCP with \( \ell < z^* < u \) is it guaranteed that \( F(z^*) = 0 \).

Internally, PATHC reformulates a constrained nonlinear system of equations to an equivalent complementarity problem. The reformulation adds variables, \( y \), with the resulting problem written as:

\[ \ell \leq x \leq u \quad \perp -y \quad y \text{ free} \quad \perp F(x). \]

This is the MCP model passed on to the PATH solver.

5.33.3 PATH Options

The default options of PATH should be sufficient for most models. If desired, PATH-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

We give a list of the available PATH options along with their defaults and meaning below. Note that only the first three characters of every word are significant.

5.33.3.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>chen_lambda</td>
<td>lambda parameter for Chen-Chen-Kanzow residual</td>
<td>0.8</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>stopping criterion When the residual error is within this tolerance, convergence is declared.</td>
<td>1e-6</td>
</tr>
<tr>
<td>crash_iteration_limit</td>
<td>maximum iterations allowed in basis crash</td>
<td>50</td>
</tr>
<tr>
<td>crashMeritFunction</td>
<td>merit function used in crash method normal: Use the normal map fischer: Use the Fischer function</td>
<td>fischer</td>
</tr>
<tr>
<td>crashMethod</td>
<td>pnewton or none pnewton: Use projected Newton method none</td>
<td>pnewton</td>
</tr>
<tr>
<td>crash_minimum_dimension</td>
<td>minimum problem dimension to perform crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_nbchange_limit</td>
<td>limit on crash iterations without basis change The basis crashing procedure is stopped if this many successive stagnant iterations occur.</td>
<td>1</td>
</tr>
<tr>
<td>crash_perturb</td>
<td>perturb the problem using pnewton crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_searchtype</td>
<td>search type to use in the crash method line: Use a linesearch arc: Use an arcsearch</td>
<td>line</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>cumulative_iteration_limit</td>
<td>maximum minor iterations allowed</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>factorization_library_name</td>
<td>name of factorization library</td>
<td></td>
</tr>
<tr>
<td>factorization_method</td>
<td>basis package to use</td>
<td>lusol</td>
</tr>
<tr>
<td></td>
<td>lusol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blu,blu_lusol</td>
<td></td>
</tr>
<tr>
<td></td>
<td>umfpack</td>
<td></td>
</tr>
<tr>
<td>gradient_searchtype</td>
<td>search type to use on a gradient step</td>
<td>arc</td>
</tr>
<tr>
<td></td>
<td>line: Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc: Use an arcssearch</td>
<td></td>
</tr>
<tr>
<td>gradient_step_limit</td>
<td>gradient steps allowed before restarting</td>
<td>5</td>
</tr>
<tr>
<td>interrupt_limit</td>
<td>ctrl-C's required before a hard kill of the solver</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>major_iteration_limit</td>
<td>maximum major iterations allowed</td>
<td>500</td>
</tr>
<tr>
<td>merit_function</td>
<td>merit function to use (normal or fischer)</td>
<td>fischer</td>
</tr>
<tr>
<td></td>
<td>normal: Use the normal map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fischer: Use the Fischer function</td>
<td></td>
</tr>
<tr>
<td>minor_iteration_limit</td>
<td>minor iterations allowed in each major iteration:</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>default MIN(2n,iterlim)</td>
<td></td>
</tr>
<tr>
<td>nms</td>
<td>allow line searching, watch-dogging, and nonmonotone descent</td>
<td>1</td>
</tr>
<tr>
<td>nms_initial_reference_factor</td>
<td>controls size of initial reference value</td>
<td>20</td>
</tr>
<tr>
<td>nms_maximum_watchdogs</td>
<td>maximum number of watchdog steps allowed</td>
<td>5</td>
</tr>
<tr>
<td>nms_memory_size</td>
<td>number of reference values kept</td>
<td>10</td>
</tr>
<tr>
<td>nms_mstep_frequency</td>
<td>frequency at which m-steps are performed</td>
<td>10</td>
</tr>
<tr>
<td>option_file</td>
<td>name of option file for PATHLIB to read</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>If specified, the PATH optimizer will read this file using its internal reader, after the usual options processing is done.</td>
<td></td>
</tr>
<tr>
<td>preprocess</td>
<td>turns preprocessing on/off</td>
<td>1</td>
</tr>
<tr>
<td>proximal_perturbation</td>
<td>initial perturbation</td>
<td>0</td>
</tr>
<tr>
<td>time_limit</td>
<td>number of seconds algorithm is allowed to run</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>lemke_rank_deficiency_iterations</td>
<td>number of attempts made to fix rank-deficient basis during Lemke start</td>
<td>10</td>
</tr>
<tr>
<td>lemke_start</td>
<td>frequency of lemke starts</td>
<td>automatic</td>
</tr>
<tr>
<td></td>
<td>always: Use a Lemke start for each LCP subproblem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>automatic: Determined by algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first: Use a Lemke start for the first LCP subproblem</td>
<td></td>
</tr>
<tr>
<td>lemke_start_type</td>
<td>type of lemke start</td>
<td>slack</td>
</tr>
<tr>
<td></td>
<td>advanced: Start Lemke method using an advanced basis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>slack: Start Lemke method using an all-slack basis</td>
<td></td>
</tr>
</tbody>
</table>
## 5.33.3.2 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>output_crash_iterations</code></td>
<td>output information on crash iterations</td>
<td>1</td>
</tr>
<tr>
<td><code>output_crash_iterations_frequency</code></td>
<td>frequency at which crash iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ( {1, \ldots, \infty} )</td>
<td></td>
</tr>
<tr>
<td><code>output_errors</code></td>
<td>output error messages</td>
<td>1</td>
</tr>
<tr>
<td><code>output_final_degeneracy_statistics</code></td>
<td>print information regarding degeneracy at the solution</td>
<td>0</td>
</tr>
<tr>
<td><code>output_final_point</code></td>
<td>output final point returned from PATH</td>
<td>0</td>
</tr>
<tr>
<td><code>output_final_point_statistics</code></td>
<td>output information about the point, function, and Jacobian at the final point</td>
<td>1</td>
</tr>
<tr>
<td><code>output_final_scaling_statistics</code></td>
<td>display matrix norms on the Jacobian at the final point</td>
<td>0</td>
</tr>
<tr>
<td><code>output_final_statistics</code></td>
<td>output evaluation of available merit functions at the final point</td>
<td>1</td>
</tr>
<tr>
<td><code>output_final_summary</code></td>
<td>output summary information</td>
<td>1</td>
</tr>
<tr>
<td><code>output_initial_point</code></td>
<td>output initial point given to PATH</td>
<td>0</td>
</tr>
<tr>
<td><code>output_initial_point_statistics</code></td>
<td>output information about the point, function, and Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td><code>output_initial_scaling_statistics</code></td>
<td>display matrix norms on the Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td><code>output_initial_statistics</code></td>
<td>output evaluation of available merit functions at the initial point</td>
<td>0</td>
</tr>
<tr>
<td><code>output_linear_model</code></td>
<td>output linear model at each major iteration</td>
<td>0</td>
</tr>
<tr>
<td><code>output_major_iterations</code></td>
<td>output information on major iterations</td>
<td>1</td>
</tr>
<tr>
<td><code>output_major_iterations_frequency</code></td>
<td>frequency at which major iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ( {1, \ldots, \infty} )</td>
<td></td>
</tr>
<tr>
<td><code>output_maximum_zero_listing</code></td>
<td>limits zero columns reported to listing file</td>
<td>1000</td>
</tr>
<tr>
<td><code>output_maximum_zero_log</code></td>
<td>limits zero columns reported to log file</td>
<td>10</td>
</tr>
<tr>
<td><code>output_minor_iterations</code></td>
<td>output information on minor iterations</td>
<td>1</td>
</tr>
<tr>
<td><code>output_minor_iterations_frequency</code></td>
<td>frequency at which minor iteration log is printed</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: ( {1, \ldots, \infty} )</td>
<td></td>
</tr>
<tr>
<td><code>output_options</code></td>
<td>output all options and their values</td>
<td>0</td>
</tr>
<tr>
<td><code>output</code></td>
<td>turns all output off or on</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>If output is first turned off, selected parts can be turned back on using specific output options.</td>
<td></td>
</tr>
<tr>
<td><code>output_preprocess_level</code></td>
<td>control output of preprocessing information</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-1, \ldots, \infty} )</td>
<td></td>
</tr>
<tr>
<td><code>output_restart_log</code></td>
<td>output options during restarts</td>
<td>1</td>
</tr>
<tr>
<td><code>output_time</code></td>
<td>output breakdown of where time is spent</td>
<td>0</td>
</tr>
<tr>
<td><code>output_warnings</code></td>
<td>output warning messages</td>
<td>0</td>
</tr>
</tbody>
</table>
GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the `solve` statement:

```plaintext
option iterlim = 2000;
transport.iterlim = 2000;
```

Problems with a singular basis matrix can be overcome by using the `proximal_perturbation` option [26], and linearly dependent columns can be output with the `output_factorization_singularity` option. For more information on singularities, we refer the reader to Section Advanced Topics.

As a special case, PATH can emulate Lemke’s method [39], [117] for LCP with the following options:

```plaintext
crash_method none
crash_perturb no
major_iteration_limit 1
lemke_start first
nms no
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [101], [132], [131] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

```plaintext
crash_method none
crash_perturb no
lemke_start always
nms_initial_reference_factor 1
nms_memory size 1
nms_mstep_frequency 1
nms_searchtype line
merit_function normal
```

Note that `nms_memory size 1` and `nms_initial_reference_factor 1` turn off the nonmonotone linesearch, while `nms_mstep_frequency 1` turns off watchdogging [35]. `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, while `merit_function normal` tell PATH to use the normal map for calculating the residual.

### 5.33.4 Advanced Topics

This chapter discusses some of the difficulties encountered when dealing with complementarity problems. We start off with a very formal definition of a complementarity problem (similar to the one given previously) which is used in later sections on merit functions and ill-defined, poorly-scaled, and singular models.

#### 5.33.4.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, $F : D \to \mathbb{R}^n$ where $D \subseteq \mathbb{R}^n$ is the domain of $F$, and possibly infinite lower and upper bounds, $\ell$ and $u$. Let $C := \{ x \in \mathbb{R}^n \mid \ell \leq x \leq u \}$, a Cartesian product of closed (possibly infinite) intervals. The problem is given as

$$MCP : \text{find } x \in C \cap D \text{ s.t. } \langle F(x), y - x \rangle \geq 0, \forall y \in C.$$

This formulation is a special case of the variational inequality problem defined by $F$ and a (nonempty, closed, convex) set $C$. Special choices of $\ell$ and $u$ lead to the familiar cases of a system of nonlinear equations

$$F(x) = 0$$

(generated by $\ell \equiv -\infty, u \equiv +\infty$) and the nonlinear complementarity problem

$$0 \leq x \perp F(x) \geq 0$$

(generated using $\ell \equiv 0, u \equiv +\infty$).
5.3.4.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

5.3.4.2.1 Merit Functions

A solver for complementarity problems typically employs a merit function to indicate the closeness of the current iterate to the solution set. The merit function is zero at a solution to the original problem and strictly positive otherwise. Numerically, an algorithm terminates when the merit function is approximately equal to zero, thus possibly introducing spurious "solutions".

The modeler needs to be able to determine with some reasonable degree of accuracy whether the algorithm terminated at solution or if it simply obtained a point satisfying the desired tolerances that is not close to the solution set. For complementarity problems, we can provide several indicators with different characteristics to help make such a determination. If one of the indicators is not close to zero, then there is some evidence that the algorithm has not found a solution. We note that if all of the indicators are close to zero, we are reasonably sure we have found a solution. However, the modeler has the final responsibility to evaluate the "solution" and check that it makes sense for their application.

For the NCP, a standard merit function is
\[
\|(-x)_+, (-F(x))_+, [(x_i)_+(F_i(x))_+]\|
\]
with the first two terms measuring the infeasibility of the current point and the last term indicating the complementarity error. In this expression, we use \((-\cdot)_+)\) to represent the Euclidean projection of \(x\) onto the nonnegative orthant, that is \((-x)_+ = \max(x, 0)\). For the more general MCP, we can define a similar function:
\[
\left\| x - \pi(x), \left( \frac{x_i - \ell_i}{\|\ell_i\| + 1} \right)_+ (F_i(x))_+ \right\|
\]
where \(\pi(x)\) represents the Euclidean projection of \(x\) onto \(C\). We can see that if we have an NCP, the function is exactly the one previously given and for nonlinear systems of equations, this becomes \(\|F(x)\|\).

There are several reformulations of the MCP as a system of nonlinear (nonsmooth) equations for which the corresponding residual is a natural merit function. Some of these are as follows:

- **Generalized Minimum Map:** \(x - \pi(x - F(x))\)
- **Normal Map:** \(F(\pi(y)) + y - \pi(y)\)
- **Fischer Function:** \(\Phi(x)\), where \(\Phi_i(x) := \phi(x_i, F_i(x))\) with
  \[
  \phi(a, b) := \sqrt{a^2 + b^2} - a - b.
  \]
  Note that \(\phi(a, b) = 0\) if and only if \(0 \leq a \perp b \geq 0\). A straightforward extension of \(\Phi\) to the MCP format is given for example in [61].

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map \(\min(x, F(x))\). Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [185]. Figure 10 in the subsequent section plots all of these merit functions for the ill-defined example discussed therein and highlights the differences between them.

The squared norm of \(\Phi\), namely \(\Psi(x) := \frac{1}{2} \sum \phi(x_i, F_i)^2\), is continuously differentiable on \(\mathbb{R}^n\) provided \(F\) itself is. Therefore, the first order optimality conditions for the unconstrained minimization of \(\Psi(x)\), namely \(\nabla \Psi(x) = 0\) give another indication as to whether the point under consideration is a solution of MCP.

The merit functions and the information PATH provides at the solution can be useful for diagnostic purposes. By default, PATH 4.x returns the best point with respect to the merit function because this iterate likely provides better information to the modeler. As detailed in Section PATH Options, the default merit function in PATH 4.x is the Fischer function. To change this behavior the `merit_function` option can be used.
5.33.4.2.2 Crashing Method  The crashing technique [47] is used to quickly identify an active set from the user-supplied starting point. At this time, a proximal perturbation scheme [24], [25] is used to overcome problems with a singular basis matrix. The proximal perturbation is introduced in the crash method, when the matrix factored is determined to be singular. The value of the perturbation is based on the current merit function value.

Even if the crash method is turned off, for example via the option crash_method none, perturbation can be added. This is determined by factoring the matrix that crash would have initially formed. This behavior is extremely useful for introducing a perturbation for singular models. It can be turned off by issuing the option crash_perturb no.

5.33.4.2.3 Nonmontone Searches  The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [90], [91], [56]. In this case we define a reference value, \( R^k \), and we use this value to test for sufficient decrease:

\[
\Psi(x^k + t_k d^k) \leq R^k + t_k \nabla \Psi(x^k)^T d^k.
\]

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

We now need to detail our choice of the reference value. We begin by letting \( \{M_1, \ldots, M_m\} \) be a finite set of values initialized to \( \kappa \Psi(x^0) \), where \( \kappa \) is used to determine the initial set of acceptable merit function values.

The value of \( \kappa \) defaults to 1 in the code and can be modified with the nms_initial_reference_factor option; \( \kappa = 1 \) indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of \( \{M_1, \ldots, M_m\} \) (where the code by default uses \( m = 10 \)), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that \( d^k \) is the Newton direction, we define \( i_0 = \arg\max M_i \) and \( R^k = M_{i_0} \). After the nonmonotone linesearch rule above finds \( t_k \), we update the memory so that \( M_{i_0} = \Psi(x^k + t_k d^k) \), i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let \( x^k = x^{\text{best}} \) where \( x^{\text{best}} \) is the point with the absolute best merit function value encountered so far. We then recalculate \( d^k = -\nabla \Psi(x^k) \) using the best point and let \( R^k = \Psi(x^k) \). That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set \( M_i = \Psi(x^k + t_k d^k) \) for all \( i \in [1, \ldots, m] \). This prevents future iterates from returning to the same problem area.

A watchdog strategy [35] is also available for use in the code. The method employed allows steps to be accepted when they are "close" to the current iterate. Nonmonotonic decrease is enforced every \( m \) iterations, where \( m \) is set by the nms_mstep_frequency option.

5.33.4.2.4 Linear Complementarity Problems  PATH solves a linear complementarity problem at each major iteration. Let \( M \in \mathbb{R}^{n \times n}, q \in \mathbb{R}^n, \) and \( B = [l, u] \) be given. \((z, w, v)\) solves the linear mixed complementarity problem defined by \( M, q, \) and \( B \) if and only if it satisfies the following constrained system of equations:

\[
Mz - w + v + q = 0 \tag{4}
\]

\[
w^T(z - l) = 0 \tag{5}
\]

\[
v^T(u - z) = 0 \tag{6}
\]
where $x + \infty = \infty$ for all $x \in \mathbb{R}$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (4) - (6) is called a complementary triple.

The objective of the linear model solver is to construct a path from a given complementary triple $(\hat{z}, \hat{w}, \hat{v})$ to a solution $(\bar{z}, \bar{w}, \bar{v})$. The algorithm used to solve the linear problem is identical to that given in [44]; however, artificial variables are incorporated into the model. The augmented system is then:

$$Mz - w + v + Da + \frac{(1 - t)}{s}(sr) + q = 0 \quad (8)$$

$$w^T(z - l) = 0 \quad (9)$$

$$v^T(u - z) = 0 \quad (10)$$

$$z \in \mathbb{B}, w \in \mathbb{R}_+^n, v \in \mathbb{R}_+^n, a \equiv 0, t \in [0, 1] \quad (11)$$

where $r$ is the residual, $t$ is the path parameter, and $a$ is a vector of artificial variables. The residual is scaled by $s$ to improve numerical stability.

The addition of artificial variables enables us to construct an initial invertible basis consistent with the given starting point even under rank deficiency. The procedure consists of two parts: constructing an initial guess as to the basis and then recovering from rank deficiency to obtain an invertible basis. The crash technique gives a good approximation to the active set. The first phase of the algorithm uses this information to construct a basis by partitioning the variables into three sets:

1. $W = \{ i \in \{1, \ldots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0 \}$
2. $V = \{ i \in \{1, \ldots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{v}_i > 0 \}$
3. $Z = \{1, \ldots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \ldots, n\}$. Using the above guess, we can recover an invertible basis consistent with the starting point by defining $D$ appropriately. The technique relies upon the factorization to indicate the linearly dependent rows and columns of the basis matrix. Some of the variables may be nonbasic, but not at their bounds. For such variables, the corresponding artificial will be basic.

We use a modified version of EXPAND [83] to perform the ratio test. Variables are prioritized as follows:

1. $t$ leaving at its upper bound.
2. Any artificial variable.
3. Any $z$, $w$, or $v$ variable.

If a choice as to the leaving variable can be made while maintaining numerical stability and sparsity, we choose the variable with the highest priority (lowest number above).

When an artificial variable leaves the basis and a $z$-type variable enters, we have the choice of either increasing or decreasing that entering variable because it is nonbasic but not at a bound. The determination is made such that $t$ increases and stability is preserved.

If the code is forced to use a ray start at each iteration (lemke_start always), then the code carries out Lemke’s method, which is known [39] not to cycle. However, by default, we use a regular start to
guarantee that the generated path emanates from the current iterate. Under appropriate conditions, this
guarantees a decrease in the nonlinear residual. However, it is then possible for the pivot sequence in the
linear model to cycle. To prevent this undesirable outcome, we attempt to detect the formation of a cycle
with the heuristic that if a variable enters the basis more than a given number of times, we are cycling.
The number of times the variable has entered is reset whenever \( t \) increases beyond its previous maximum
or an artificial variable leaves the basis. If cycling is detected, we terminate the linear solver at the largest
value of \( t \) and return this point.

Another heuristic is added when the linear code terminates on a ray. The returned point in this case is
not the base of the ray. We move a slight distance up the ray and return this new point. If we fail to solve
the linear subproblem five times in a row, a Lemke ray start will be performed in an attempt to solve the
linear subproblem. Computational experience has shown this to be an effective heuristic and generally
results in solving the linear model. Using a Lemke ray start is not the default mode, since typically many
more pivots are required.

For times when a Lemke start is actually used in the code, an advanced ray can be used. We basically
choose the ”closest” extreme point of the polytope and choose a ray in the interior of the normal cone at
this point. This helps to reduce the number of pivots required. However, this can fail when the basis
pertaining to the cell is not invertible. We then revert to the Lemke start.

Since the EXPAND pivot rules are used, some of the variables may be nonbasic, but slightly infeasible,
as the solution. Whenever the linear code finishes, the nonbasic variables are put at their bounds and
the basic variables are recomputed using the current factorization. This procedure helps to find the best
possible solution to the linear system.

The resulting linear solver as modified above is robust and has the desired property that we start from
\((\hat{z}, \hat{w}, \hat{v})\) and construct a path to a solution.

5.33.4.2.5 Other Features Some other heuristics are incorporated into the code. During the first
iteration, if the linear solver fails to find a Newton point, a Lemke start is used. Furthermore, under
repeated failures during the linear solve, a Lemke start will be attempted. A gradient step can also be
used when we fail repeatedly.

The proximal perturbation is reduced at each major iteration. However, when numerical difficulties are
encountered, it will be increased to a fraction of the current merit function value. These are determined
when the linear solver returns a Reset or Singular status.

Spacer steps are taken every major iteration, in which the iterate is chosen to be the best point for the
normal map. The corresponding basis passed into the Lemke code is also updated.

Scaling is done based on the diagonal of the matrix passed into the linear solver.

We finally note, that if the merit function fails to show sufficient decrease over the last 100 iterates, a
restart will be performed, as this indicates we are close to a stationary point.

5.33.4.3 Difficult Models

5.33.4.3.1 Ill-Defined Models A problem can be ill-defined for several different reasons. We
concentrate on the following particular cases. We will call \( F \) well-defined at \( \bar{x} \in C \) if \( \bar{x} \in D \) and ill-
defined at \( \bar{x} \) otherwise. Furthermore, we define \( F \) to be well-defined near \( \bar{x} \in C \) if there exists an open
neighborhood of \( \bar{x}, \mathcal{N}(\bar{x}) \), such that \( C \cap \mathcal{N}(\bar{x}) \subseteq D \). By saying the function is well-defined near \( \bar{x} \), we are
simply stating that \( F \) is defined for all \( x \in C \) sufficiently close to \( \bar{x} \). A function not well-defined near \( \bar{x} \) is
termned ill-defined near \( \bar{x} \).

We will say that \( F \) has a well-defined Jacobian at \( \bar{x} \in C \) if there exists an open neighborhood of \( \bar{x}, \mathcal{N}(\bar{x}) \), such that \( \mathcal{N}(\bar{x}) \subseteq D \) and \( F \) is continuously differentiable on \( \mathcal{N}(\bar{x}) \). Otherwise the function has
an ill-defined Jacobian at $\bar{x}$. We note that a well-defined Jacobian at $\bar{x}$ implies that the MCP has a well-defined function near $\bar{x}$, but the converse is not true.

PATH uses both function and Jacobian information in its attempt to solve the MCP. Therefore, both of these definitions are relevant. We discuss cases where the function and Jacobian are ill-defined in the next two subsections. We illustrate uses for the merit function information and final point statistics within the context of these problems.

**Function Undefined** We begin with a one-dimensional problem for which $F$ is ill-defined at $x = 0$ as follows:

$$0 \leq x \perp \frac{1}{x} \geq 0.$$

Here $x$ must be strictly positive because $\frac{1}{x}$ is undefined at $x = 0$. This condition implies that $F(x)$ must be equal to zero. Since $F(x)$ is strictly positive for all $x$ strictly positive, this problem has no solution.

We are able to perform this analysis because the dimension of the problem is small. Preprocessing linear problems can be done by the solver in an attempt to detect obviously inconsistent problems, reduce problem size, and identify active components at the solution. Similar processing can be done for nonlinear models, but the analysis becomes more difficult to perform. Currently, PATH only checks the consistency of the bounds and removes fixed variables and the corresponding complementary equations from the model.

A modeler would likely not know a priori that a problem has no solution and would thus attempt to formulate and solve it. GAMS code for the model above is provided in Figure 8. We must specify an initial value for $x$ in the code. If we were to not provide one, GAMS would use $x = 0$ as the default value, notice that $F$ is undefined at the initial point, and terminate before giving the problem to PATH. The error message indicates that the function $\frac{1}{x}$ is ill-defined at $x = 0$, but does not imply that the corresponding MCP problem has no solution.

**Figure 8:** GAMS Code for Ill-Defined Function

```gams
positive variable x;
equations F;
F.. 1 / x =g= 0;
model simple / F.x /;
x.l = 1e-6;
solve simple using mcp;
```

After setting the starting point, GAMS generates the model, and PATH proceeds to "solve" it. A portion of the output relating statistics about the solution is given in Figure 9. PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in Section Merit Functions at the final point. The Normal Map merit function, and to a lesser extent, the complementarity error, indicate that the "solution" found does not necessarily solve the MCP.

**Figure 9:** PATH Output for Ill-Defined Function

<table>
<thead>
<tr>
<th>FINAL STATISTICS</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inf-Norm of Complementarity</td>
<td>1.0000e+00</td>
</tr>
<tr>
<td>Inf-Norm of Normal Map</td>
<td>1.1181e+16</td>
</tr>
<tr>
<td>Inf-Norm of Minimum Map</td>
<td>8.9441e-17</td>
</tr>
<tr>
<td>Inf-Norm of Fischer Function</td>
<td>8.9441e-17</td>
</tr>
<tr>
<td>Inf-Norm of Grad Fischer Fcn.</td>
<td>8.9441e-17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FINAL POINT STATISTICS</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum of X</td>
<td>8.9441e-17</td>
</tr>
<tr>
<td>Maximum of F</td>
<td>1.1181e+16</td>
</tr>
<tr>
<td>Maximum of Grad F</td>
<td>1.2501e+32</td>
</tr>
</tbody>
</table>
To indicate the difference between the merit functions, Figure 10 plots them all for this simple example. We note that as \( x \) approaches positive infinity, numerically, we are at a solution to the problem with respect to all of the merit functions except for the complementarity error, which remains equal to one. As \( x \) approaches zero, the merit functions diverge, also indicating that \( x = 0 \) is not a solution.

The natural residual and Fischer function tend toward 0 as \( x \downarrow 0 \). From these measures, we might think \( x = 0 \) is the solution. However, as previously remarked \( F \) is ill-defined at \( x = 0 \). \( F \) and \( \nabla F \) become very large, indicating that the function (and Jacobian) might not be well-defined. We might be tempted to conclude that if one of the merit function indicators is not close to zero, then we have not found a solution. This conclusion is not always warranted. When one of the indicators is non-zero, we have reservations about the solution, but we cannot eliminate the possibility that we are actually close to a solution. If we slightly perturb the original problem to

\[
0 \leq x \perp \frac{1}{x + \epsilon} \geq 0
\]

for a fixed \( \epsilon > 0 \), the function is well-defined over \( C = \mathbb{R}_+^n \) and has a unique solution at \( x = 0 \). In this case, by starting at \( x > 0 \) and sufficiently small, all of the merit functions, with the exception of the Normal Map, indicate that we have solved the problem as is shown by the output in Figure 11 for \( \epsilon = 1 \times 10^{-6} \) and \( x = 1 \times 10^{-20} \).

**Figure 11: PATH Output for Well-Defined Function**

---

**FINAL STATISTICS**

- Inf-Norm of Complementarity . . 1.0000e-14 eqn: (G)
- Inf-Norm of Normal Map . . . . 1.0000e+06 eqn: (G)
- Inf-Norm of Minimum Map . . . . 1.0000e-20 eqn: (G)
- Inf-Norm of Fischer Function . . 1.0000e-20 eqn: (G)
- Inf-Norm of Grad Fischer Fcn . . 1.0000e-20 eqn: (G)
In this case, the Normal Map is quite large and we might think that the function and Jacobian are undefined. When only the normal map is non-zero, we may have just mis-identified the optimal basis. By setting the merit function normal option, we can resolve the problem, identify the correct basis, and solve the problem with all indicators being close to zero. This example illustrates the point that all of these tests are not infallible. The modeler still needs to do some detective work to determine if they have found a solution or if the algorithm is converging to a point where the function is ill-defined.

**Jacobian Undefined** Since PATH uses a Newton-like method to solve the problem, it also needs the Jacobian of $F$ to be well-defined. One model for which the function is well-defined over $C$, but for which the Jacobian is undefined at the solution is: $0 \leq x^\perp - \sqrt{x} \geq 0$. This model has a unique solution at $x = 0$.

Using PATH and starting from the point $x = 1 \times 10^{-14}$, PATH generates the output given in Figure 12.

**Figure 12:** PATH Output for Ill-Defined Jacobian

---

We can see that the gradient of the Fischer Function is nonzero and the Jacobian is beginning to become large. These conditions indicate that the Jacobian is undefined at the solution. It is therefore important for a modeler to inspect the given output to guard against such problems.

If we start from $x = 0$, PATH correctly informs us that we are at the solution. The large value reported for the Jacobian is an indication that the Jacobian is undefined.

### 5.33.4.3.2 Poorly Scaled Models

Well-defined problems can still have various numerical problems that impede the algorithm's convergence. One particular problem is a badly scaled Jacobian. In such cases, we can obtain a poor "Newton" direction because of numerical problems introduced in the linear algebra performed. This problem can also lead the code to a point from which it cannot recover.

The final model given to the solver should be scaled such that we avoid numerical difficulties in the linear algebra. The output provided by PATH can be used to iteratively refine the model so that we eventually end up with a well-scaled problem. We note that we only calculate our scaling statistics at the starting point provided. For nonlinear problems these statistics may not be indicative of the overall scaling of the model. Model specific knowledge is very important when we have a nonlinear problem because it can be used to appropriately scale the model to achieve a desired result.

We look at the titan.gms model in MCPLIB, that has some scaling problems. The relevant output from PATH for the original code is given in Figure 13.

**Figure 13:** PATH Output - Poorly Scaled Model
INITIAL POINT STATISTICS

Maximum of X. . . . . . . . . . . 4.1279e+06 var: (w.29)
Maximum of F. . . . . . . . . . . 2.2516e+00 eqn: (a1.33)
Maximum of Grad F . . . . . . . . 6.7753e+06 eqn: (a1.29)

INITIAL JACOBIAN NORM STATISTICS

Maximum Row Norm. . . . . . . . . 9.4504e+06 eqn: (a2.29)
Minimum Row Norm. . . . . . . . . 2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . . . 9.4504e+06 var: (x1.29)
Minimum Column Norm . . . . . . . 1.3840e-03 var: (w.10)

The maximum row norm is defined as
\[ \max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}| \]
and the minimum row norm is
\[ \min_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}| . \]

Similar definitions are used for the column norm. The norm numbers for this particular example are not extremely large, but we can nevertheless improve the scaling. We first decided to reduce the magnitude of the a1 block of equations as indicated by PATH. Using the GAMS modeling language, we can scale particular equations and variables using the .scale attribute. To turn the scaling on for the model we use the .scaleopt model attribute. After scaling the a1 block, we re-ran PATH and found an additional block of equations that also needed scaling, a2. We also scaled some of the variables, g and w. The code added to the model follows:

```gams
  titan.scaleopt = 1;
  a1.scale(i) = 1000;
  a2.scale(i) = 1000;
  g.scale(i) = 1/1000;
  w.scale(i) = 100000;
```

By scaling these equation and variable blocks, we have improved the model scaling. The statistics for the manually scaled model are given in Figure 14.

Figure 14: PATH Output - Well-Scaled Model

INITIAL POINT STATISTICS

Maximum of X. . . . . . . . . . . 1.0750e+03 var: (x1.49)
Maximum of F. . . . . . . . . . . 3.9829e-01 eqn: (g.10)
Maximum of Grad F . . . . . . . . 6.7753e+06 eqn: (a1.29)

INITIAL JACOBIAN NORM STATISTICS

Maximum Row Norm. . . . . . . . . 9.4524e+06 eqn: (a2.29)
Minimum Row Norm. . . . . . . . . 2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . . . 9.4904e+03 var: (x2.29)
Minimum Column Norm . . . . . . . 1.3840e-01 var: (w.10)

For this particular problem PATH cannot solve the unscaled model, while it can find a solution to the scaled model. Using the scaling features of the GAMS language and the information provided by PATH we are able to remove some of the problem’s difficulty and obtain better performance from PATH.

It is possible to get even more information on initial point scaling by inspecting the GAMS listing file. The equation row listing gives the values of all the entries of the Jacobian at the starting point. The row norms generated by PATH give good pointers for starting to use the row listing.

Not all of the numerical problems are directly attributable to poorly scaled models. Problems for which the Jacobian of the active constraints is singular or nearly singular can also cause numerical difficulty as illustrated next.
5.33.4.3.3 **Singular Models**  Assuming that the problem is well-defined and properly scaled, we can still have a Jacobian for which the active constraints are singular or nearly singular (i.e. it is ill-conditioned). When problems are singular or nearly singular, we are also likely to have numerical problems. As a result the "Newton" direction obtained from the linear problem solver can be very bad. In PATH, we can use proximal perturbation or add artificial variables to attempt to remove the singularity problems from the model. However, it is most often beneficial for solver robustness to remove singularities if possible.

The easiest problems to detect are those for which the Jacobian has zero rows and columns. A simple problem for which we have zero rows and columns is:

\[-2 \leq x \leq 2 \quad \perp \quad -x^2 + 1.\]

Note that the Jacobian, \(-2x\), is non-singular at all three solutions, but singular at the point \(x = 0\). Output from PATH on this model starting at \(x = 0\) is given in Figure 15.

**Figure 15:** PATH Output - Zero Rows and Columns

```
INITIAL POINT STATISTICS
Zero column of order . . . . . . 0.0000e+00 var: (X)
Zero row of order . . . . . . . 0.0000e+00 eqn: (F)
Total zero columns . . . . . . . 1
Total zero rows . . . . . . . . . 1
Maximum of F . . . . . . . . . . 1.0000e+00 eqn: (F)
Maximum of Grad F . . . . . . . 0.0000e+00 eqn: (F)
                      var: (X)
```

We display in the code the variables and equations for which the row/column in the Jacobian is close to zero. These situations are problematic and for nonlinear problems likely stem from the modeler providing an inappropriate starting point or fixing some variables resulting in some equations becoming constant. We note that the solver may perform well in the presence of zero rows and/or columns, but the modeler should make sure that these are what was intended.

Singularities in the model can also be detected by the linear solver. This in itself is a hard problem and prone to error. For matrices which are poorly scaled, we can incorrectly identify "linearly dependent" rows because of numerical problems. Setting `output_factorization_singularities yes` in an options file will inform the user which equations the linear solver thinks are linearly dependent. Typically, singularity does not cause a lot of problems and the algorithm can handle the situation appropriately. However, an excessive number of singularities are cause for concern. A further indication of possible singularities at the solution is a lack of quadratic convergence to the solution.

5.33.5 **Case Study: Von Thunen Land Model**

We now turn our attention towards using the diagnostic information provided by PATH to improve an actual model. The Von Thunen land model is a problem recognized in the mathematical programming literature for its computational difficulty. We attempt to understand more carefully the facets of the problem that make it difficult to solve. This will enable us to outline and identify these problems and furthermore to extend the model to a more realistic and computationally more tractable form.
5.33.5.1 Classical Model

The problem is cast in the Arrow-Debreu framework as an equilibrium problem. The basic model is a closed economy consisting of three economic agents, a landowner, a worker and a porter. There is a central market, around which concentric regions of land are located. Since the produced goods have to be delivered to the market, this is an example of a spatial price equilibrium. The key variables of the model are the prices of commodities, land, labour and transport. Given these prices, it is assumed that the agents demand certain amounts of the commodities, which are supplied so as to maximize profit in each sector. Walras’ law is then a consequence of the assumed competitive paradigm, namely that supply will equal demand in the equilibrium state.

We now describe the problems that the consumers and the producers face. We first look at consumption and derive a demand function for each of the consumer agents in the economy. Each of these agents has a utility function, that they wish to maximize subject to their budgetary constraints. As is typical in such problems, the utility function is assumed to be Cobb-Douglas:

\[ u_a(d) = \prod_c d_c^{\alpha_{c,a}} \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1, \]

where the \( \alpha_{c,a} \) are given parameters dependent only on the agent. For each agent \( a \), the variables \( d_c \) represent quantities of the desired commodities \( c \). In the Von Thunen model, the goods are wheat, rice, corn and barley. The agent’s endowments determine their budgetary constraint as follows. Given current market prices, an agent’s wealth is the value of the initial endowment of goods at those prices. The agent’s problem is therefore

\[ \max u_a(d) \text{ subject to } (p, d) \leq (p, e_a), d \geq 0, \]

where \( e_a \) is the endowment bundle for agent \( a \). A closed form solution, corresponding to demand from agent \( a \) for commodity \( c \) is thus

\[ d_{c,a}(p) := \frac{\alpha_{c,a}[p, e_a]}{p_c}. \]

Note that this assumes the prices of the commodities \( p_c \) are positive.

The supply side of the economy is similar. The worker earns a wage \( w_L \) for his labour input. The land is distributed around the market in rings with a rental rate \( w_r \) associated with each ring \( r \) of land. The area of land \( a_r \) in each ring is an increasing function of \( r \). The model assumes that labour and land are substitutable via a constant elasticities of substitution (CES) function.

Consider the production \( x_{c,r} \) of commodity \( c \) in region \( r \). In order to maximize profit (or minimize costs), the labour \( y_L \) and land use \( y_r \) solve

\[ \min w_L y_L + w_r y_r \text{ subject to } \phi_c y_c^{\beta_c} y_r^{1-\beta_c} \geq x_{c,r}, \quad y_L, y_r \geq 0, \]

(12)

where \( \phi_c \) is a given cost function scale parameter, and \( \beta_c \in [0, 1] \) is the share parameter. The technology constraint is precisely the CES function allowing a suitable mix of labour and land use. Again, a closed form solution can be calculated. For example, the demand for labour in order to produce \( x_{c,r} \) of commodity \( c \) in region \( r \) is given by

\[ x_{c,r} \left( \frac{w_L}{\phi_c} \right)^{\beta_c} \left( \frac{w_r}{1-\beta_c} \right)^{1-\beta_c}. \]

Considering all such demands, this clearly assumes the prices of inputs \( w_L, w_r \) are positive. A key point to note is that input commodity (factor) demands to produce \( x_{c,r} \) can be determined by first solving (12) for unit demand \( x_{c,r} \equiv 1 \) and then multiplying these factor demands by the actual amount desired. Let \( \bar{y}_L \) and \( \bar{y}_r \) denote the optimal solutions of (12) with \( x_{c,r} \equiv 1 \). Using this fact, the unit production cost \( \gamma_{c,r} \) for commodity \( c \) in region \( r \) can be calculated as follows:
\( \gamma_{c,r} = w_L y_L + w_r y_r \)
\( \gamma_{c,r} = w_L \left( \frac{w_L}{\phi_c w_L} \right)^{\beta_c} \left( \frac{w_r}{1 - \beta_c} \right)^{1 - \beta_c} \)
\( \gamma_{c,r} = \frac{1}{\phi_c} \left( \frac{w_L}{\beta_c} \right)^{\beta_c} \left( \frac{w_r}{1 - \beta_c} \right)^{1 - \beta_c} \).

Transportation is provided by a porter, earning a wage \( w_p \). If we denote the unit cost for transportation of commodity \( c \) by \( t_c \), then unit transportation cost to market is
\( T_{c,r}(w_p) := t_c d_r w_p \),
where \( d_r \) is the distance of region \( r \) to the market. Spatial price equilibrium arises from the consideration:
\( 0 \leq x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \geq p_c \).
This is intuitively clear; it states that commodity \( c \) will be produced in region \( r \) only if the combined cost of production and transportation equals the market price.

The above derivations assumed that the producers and consumers acted as price takers. Walras’ law is now invoked to determine the prices so that markets clear. The resulting complementarity problem is:
\( \gamma_{c,r} = \frac{1}{\phi_c} \left( \frac{w_L}{\beta_c} \right)^{\beta_c} \left( \frac{w_r}{1 - \beta_c} \right)^{1 - \beta_c} \) (13)
\( 0 \leq x_{c,r} \perp \gamma_{c,r} + T_{c,r}(w_p) \geq p_c \) (14)
\( 0 \leq w_L \perp \epsilon_L \geq \sum_{r,c} x_{c,r} \frac{\beta_c \gamma_{c,r}}{w_L} \) (15)
\( 0 \leq w_r \perp \epsilon_r \geq \sum_{c} x_{c,r} (1 - \beta_c) \gamma_{c,r} \) (16)
\( 0 \leq w_p \perp \epsilon_p \geq \sum_{r,c} t_c d_r x_{c,r} \) (17)
\( 0 \leq p_c \perp \sum_{r,c} x_{c,r} \geq \frac{\alpha_c P \epsilon_p w_p + \alpha_c L \epsilon_L w_L + \alpha_c O \sum_r w_r a_r}{p_c} \) (18)

Note that in (15), (16) and (17), the amounts of labour, land and transport are bounded from above, and hence the prices on these inputs are determined as multipliers (or shadow prices) on the corresponding constraints. The final relationship (18) in the above complementarity problem corresponds to market clearance; prices are nonnegative and can only be positive if supply equals demand. (Some modelers multiply the last inequality throughout by \( p_c \). This removes problems where \( p_c \) becomes zero, but can also introduce spurious solutions.)

The Arrow-Debreu theory guarantees that the problem is homogeneous in prices; \( (x, \lambda w, \lambda p) \) is also a solution whenever \( (x, w, p) \) solves the above. Typically this singularity in the model is removed by fixing a numeraire, that is fixing a price (for example \( w_L = 1 \)) and dropping the corresponding complementary relationship.
Unfortunately, in this formulation even after fixing a numeraire, some of the variables $p$ and $w$ may go to zero, resulting in an ill-defined problem. In the case of the Von Thunen land model, the rental price of land $w_r$ decreases as the distance to market increases, and for remote rings of land, it becomes zero. A standard modeling fix is to put artificial lower bounds on these variables. Even with this fix, the problem typically remains very hard to solve. More importantly, the homogeneity property of the prices used above to fix a numeraire no longer holds, and the corresponding complementary relationship (which was dropped from the problem) may fail to be satisfied. It therefore matters which numeraire is fixed, and many modelers run into difficulty since in many cases the solution found by a solver is invalid for the originally posed model.

In order to test our diagnostic information, we implemented a version of the above model in GAMS. The model corresponds closely to the MCPLIB model pgvon105.gms except we added more regions to make the problem even more difficult. The model file has been documented more fully, and the data rounded to improve clarity.

Our first trial was to solve the model without fixing a numeraire. In this case, PATH 4.x failed to find a solution. At the starting point, the indicators described in Section Ill-Defined Models are reasonable, and there are no zero rows/columns in the Jacobian. At the best point found, all indicators are still reasonable. However, the listing file indicates a large number of division by zero problems occurring in (16). We also note that a nonzero proximal perturbation is used in the first iteration of the crash method. This is an indication of singularities. We therefore added an option to output factorization singularities, and singularities appeared in the first iteration. At this point, we decided to fix a numeraire to see if this alleviated the problem.

We chose to fix the labour wage rate to 1. After increasing the iterations allowed to 100,000, PATH 4.x solved the problem. The statistics at the solution are cause for concern. In particular, the gradient of the Fischer function is 7 orders of magnitude larger than all the other residuals. Furthermore, the Jacobian is very large at the solution point. Looking further in the listing file, a large number of division by zero problems occur in (16).

To track down the problem further, we added an artificial lower bound on the variables $w_r$ of $10^{-5}$, that would not be active at the aforementioned solution. Resolving gave the same “solution”, but resulted in the domain errors disappearing.

Although the problem is solved, there is concern on two fronts. Firstly, the gradient of the Fischer function should go to zero at the solution. Secondly, if a modeler happens to make the artificial lower bounds on the variables a bit larger, then they become active at the solution, and hence the constraint that has been dropped by fixing the price of labour at 1 is violated at this point. Of course, the algorithm is unable to detect this problem, since it is not part of the model that is passed to it, and the corresponding output looks satisfactory.

We are therefore led to the conclusion that the model as postulated is ill-defined. The remainder of this section outlines two possible modeling techniques to overcome the difficulties with ill-defined problems of this type.

### 5.33.5.2 Intervention Pricing

The principal difficulty is the fact that the rental prices on land go to zero as proximity to the market decreases, and become zero for sufficiently remote rings. Such a property is unlikely to hold in a practical setting. Typically, a landowner has a minimum rental price (for example, land in fallow increases in value). As outlined above, a fixed lower bound on the rental price violates the well-established homogeneity property. A suggestion postulated by Professor Thomas Rutherford is to allow the landowner to intervene and “purchase-back” his land whenever the rental cost gets smaller than a certain fraction of the labour wage.

The new model adds a (homogeneous in price) constraint

$$0 \leq i_r \perp w_r \geq 0.0001 \ast w_L$$
and modifies (16) and (18) as follows:

\[
0 \leq w_r \top a_r - i_r \geq \sum_c x_{c,r}(1 - \beta_c)\gamma_{c,r} \frac{w_r}{w_L}
\]

\[
0 \leq p_c \top \sum_r x_{c,r} \geq \frac{\alpha_{c,P}PePw_p + \alpha_{c,L}eLw_L + \alpha_{c,O} \sum_r w_r (a_r - i_r)}{p_c}.
\]

Given the intervention purchase, we can now add a lower bound on \(w_r\) to avoid division by zero errors. In our model we chose \(10^{-5}\) since this will never be active at the solution and therefore will not affect the positive homogeneity. After this reformulation, PATH 4.x solves the problem. Furthermore, the gradient of the Fischer function, although slightly larger than the other residuals, is quite small, and can be made even smaller by reducing the convergence tolerance of PATH. Inspecting the listing file, the only difficulties mentioned are division by zero errors in the market clearance condition (19), that can be avoided a posteriori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

### 5.33.5.3 Nested Production and Maintenance

Another observation that can be used to overcome the land price going to zero is the fact that land typically requires some maintenance labour input to keep it usable for crop growth. Traditionally, in economics, this is carried out by providing a nested CES function as technology input to the model. The idea is that commodity \(c\) in region \(r\) is made from labour and an intermediate good, which is "maintained land". Essentially, the following production problem replaces (12):

\[
\begin{align*}
\min_{y_M, y_L, y_r, g} & \quad w_L(y_M + y_L) + w_r y_r \\
\text{subject to} & \quad y_r \geq (1 - \beta_c - \epsilon)g \\
& \quad y_M \geq \epsilon g \\
& \quad \phi_c y_L^{\beta_c} g^{-\beta_c} \geq 1, \\
& \quad y_M, y_L, y_r, g \geq 0.
\end{align*}
\]

Note that the variable \(y_M\) represents "maintenance labour" and \(g\) represents the amount of "maintained land" produced, an intermediate good. The process of generating maintained land uses a Leontief production function, namely

\[
\min(\lambda_r y_r, \lambda_M y_M) \geq g.
\]

Here \(\lambda_M = \frac{1}{2}, \epsilon\) small, corresponds to small amounts of maintenance labour, while \(\lambda_r = \frac{1}{1 - \beta_c - \epsilon}\) is chosen to calibrate the model correctly. A simple calculus exercise then generates appropriate demand and cost expressions. The resulting complementarity problem comprises (14), (17), (18) and

\[
\begin{align*}
\gamma_{c,r} &= \frac{w_r}{w_L} \left( \frac{w_L (1 - \beta_c - \epsilon)}{1 - \beta_c} \right)^{1-\beta_c} \\
0 &\leq w_L \top e_L \geq \sum_{r,c} x_{c,r} \gamma_{c,r} \left( \beta_c \frac{w_L}{w_L + w_r (1 - \beta_c - \epsilon)} + \epsilon (1 - \beta_c) \right) \\
0 &\leq w_r \top a_r \geq \sum_c x_{c,r} \gamma_{c,r} \left( \frac{1 - \beta_c}{w_L + w_r (1 - \beta_c - \epsilon)} \right)
\end{align*}
\]

After making the appropriate modifications to the model file, PATH 4.x solved the problem on defaults without any difficulties. All indicators showed the problem and solution found to be well-posed.
5.34 quadMINOS

For LP (and RMIP) models that cause difficulty for the standard solvers, quadMINOS provides considerably greater reliability and accuracy than MINOS by using quadruple-precision floating-point arithmetic. Although it is slower by a factor of 20 or 30 for cold-start runs, you can use standard MINOS or another LP solver first and let them do their best. Warm-starts with quadMINOS may give greatly improved solutions at moderate cost.

To take advantage of quadMINOS, the main requirement is to set a few runtime options to non-default values in order to request higher accuracy. For example, it is reasonable to set

```
  Feasibility tolerance 1e-15
  Optimality tolerance 1e-15
```

because IEEE Quad floating-point has nearly 34 digits of precision. For an example of using Double and Quad MINOS together, see [DQQ] in the GAMS model library.

Note

GAMS/quadMINOS can only solve LPs and RMIPs at the moment because GAMS does not provide function evaluations in quad-precision.

The documentation of MINOS also refers to quadMINOS.

5.35 SBB

5.35.1 Introduction

SBB is a GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. SBB can use all GAMS NLP solvers as subsolvers but it works best with NLP solvers that can utilize a near optimal point as a starting point like Conopt, Minos, and Snopt.

SBB supports all types of discrete variables supported by GAMS, including Binary, Integer, Semicont, Semiint, Sos1, and Sos2.

5.35.2 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option `infeasseq` and `failseq` below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option `infeasseq` below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.
5.35.3 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh. A Computational Study of Search Strategies for Mixed Integer Programming, INFORMS Journal on Computing, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option varsel). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option nodesel).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are not good candidates for search strategies which require pseudo costs (varsel 3, nodesel 3,5,6). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (varsel 3, nodesel 3,5,6) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

5.35.4 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are IterLim, ResLim, NodLim, OptCA, OptCR, OptfFile, Cheat, and CutOff. A description of all available GAMS options can be found in the GAMS User's Guide Solver related options. GAMS options PriorOpt and TryInt are also accepted by SBB.

SBB uses the var.prior information to select the fractional variable with the smallest priority during the variable selection process. SBB uses the TryInt information to set the branching direction in the B&B algorithm. At the beginning, SBB looks at the levels of the discrete variables provided by the user and if Abs(Round(x.l)-x.l) < m.TryInt, SBB will branch on that variable in the direction of Round(x.l). For example, x.l=0.9 and m.TryInt=0.2. We have Abs(Round(0.9)-0.9)=0.1 < 0.2, so when SBB decides to branch on this variable (because it is fractional, lets say with value 0.5), the node explored next will have the additional constraint x ≥ 1 (the node with x ≤ 0 will be explored later). If everything goes well (there is the chance that we end up in a different local optima in the subsolves for non-convex problems), SBB should reproduce a preset incumbent solution in a couple of nodes.

If you specify <modelname>.OptFile = 1; before the Solve statement in your GAMS model, SBB will then look for and read an option file with the name sbb.opt (see The Solver Option File for general use of solver option files). Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```
optname value
```

with one option on each line.

For example,

```
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file conopt.opt will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptnonopt</td>
<td>accepts feasible solution from subsolver</td>
<td>0</td>
</tr>
<tr>
<td>avgresmult</td>
<td>average resource multiplicator</td>
<td>5</td>
</tr>
<tr>
<td>dfsstay</td>
<td>keeps DFS node selection after solution has been found</td>
<td>0</td>
</tr>
<tr>
<td>epint</td>
<td>integer feasibility tolerance</td>
<td>1.0e-5</td>
</tr>
<tr>
<td>failseq</td>
<td>solver sequence for failed nodes</td>
<td></td>
</tr>
<tr>
<td>infeasseq</td>
<td>solver sequence for infeasible nodes</td>
<td></td>
</tr>
<tr>
<td>intsollim</td>
<td>maximum number of integer solutions</td>
<td>2100000000</td>
</tr>
<tr>
<td>loginterval</td>
<td>progress display interval</td>
<td>1</td>
</tr>
<tr>
<td>loglevel</td>
<td>level of solver display</td>
<td>1</td>
</tr>
<tr>
<td>memnodes</td>
<td>maximum number of nodes in memory</td>
<td>10000</td>
</tr>
<tr>
<td>miptrace</td>
<td>filename of MIP trace file</td>
<td></td>
</tr>
<tr>
<td>miptracenode</td>
<td>node interval when a trace record is written</td>
<td>100</td>
</tr>
<tr>
<td>miptracetimer</td>
<td>time interval when a trace record is written</td>
<td>5.0</td>
</tr>
<tr>
<td>nodesel</td>
<td>node selection strategy</td>
<td>0</td>
</tr>
<tr>
<td>printbbinfo</td>
<td>prints additional node info</td>
<td>0</td>
</tr>
<tr>
<td>rootsolver</td>
<td>solver for the root node</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for GAMS NLP solver</td>
<td>5</td>
</tr>
<tr>
<td>subiter</td>
<td>iteration limit for the subsolve</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>subres</td>
<td>resource limit for the subsolve</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>subsolver</td>
<td>solver for the subproblems</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>usercallparmfile</td>
<td>Command-line parameter include file used in GAMS command-line calls triggered by BCH</td>
<td></td>
</tr>
<tr>
<td>usergdxname</td>
<td>the name of the GDX file exported from the solver with the solution at the node</td>
<td>bchout.gdx</td>
</tr>
<tr>
<td>usergdxnameinc</td>
<td>the name of the GDX file exported from the solver with the incumbent solution</td>
<td>bchout.gdx</td>
</tr>
<tr>
<td>userheurcall</td>
<td>the GAMS command line to call the heuristic</td>
<td></td>
</tr>
<tr>
<td>userheurfst</td>
<td>calls the cut generator for the first n nodes</td>
<td>10</td>
</tr>
<tr>
<td>userheurfreq</td>
<td>determines the frequency of the cut generator model calls</td>
<td>10</td>
</tr>
<tr>
<td>userheurinterval</td>
<td>determines the interval when to apply the multiplier for the frequency of the cut generator model calls</td>
<td>100</td>
</tr>
<tr>
<td>userheurmult</td>
<td>determines the multiplier for the frequency of the cut generator model calls</td>
<td>2</td>
</tr>
<tr>
<td>userheurnewint</td>
<td>calls the heuristic if the solver found a new integer feasible solution</td>
<td>0</td>
</tr>
<tr>
<td>userheurobjfirst</td>
<td>Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement</td>
<td>50</td>
</tr>
<tr>
<td>varsel</td>
<td>variable selection strategy at each node</td>
<td>0</td>
</tr>
</tbody>
</table>
acceptnonopt (boolean): accepts feasible solution from subsolver

If this option is set to 1 and the subsolver terminates with solver status Terminated by Solver and model status Intermediate Nonoptimal SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the failseq is consulted.

Default: 0

avgresmult (integer): average resource multiplicator

Similar to subres, this option allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor avgresmult is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the failseq are called. The default multiplier avgresmult is 5. Setting avgresmult to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at least 5 seconds to solve the node. The competing option subres overwrites the automatically generated resource limit.

Default: 5

dfsstay (integer): keeps DFS node selection after solution has been found

If the node selection is a B*/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B* node selection mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting dfsstay to n instructs SBB to stay in DFS mode for another n nodes.

Default: 0

epint (real): integer feasibility tolerance

The integer infeasibility tolerance.

Range: [1e-9, 1]

Default: 1.0e-5

failseq (string): solver sequence for failed nodes

solver1[.n1] solver2[.n2] ... where solver1 is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain limit and does not return a locally optimal or locally infeasible solution. n1 is the value of optfile passed to the alternative NLP solver. If .n1 is left blank it is interpreted as zero. Similarly, solver2 is the name of a GAMS NLP solver that is used if solver1 fails, and n2 is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more solver.n pairs. You can use the same solver several times with different options files. failseq conopt conopt.2 conopt.3 means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file conopt.op2, and if it again fails, try CONOPT with options file conopt.op3. If all solver and options file combinations fail the node will be labeled ignored and the node will not be explored further. The default is to try only one solver (the rootsolver or subsolver) and to ignore nodes with a solver failure.

infeasseq (string): solver sequence for infeasible nodes
level solver1[,n1] solver2[,n2] ... The purpose of infeasseq is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node Locally Infeasible and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value level. If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that Locally Infeasible nodes are indeed infeasible and to remove them from further consideration.

intsol {integer}: maximum number of integer solutions ➞

Maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: 2100000000

loginterval {integer}: progress display interval ➞

The interval (number of nodes) for which log lines are written.

Default: 1

loglevel {integer}: level of solver display ➞

The level of log output.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>only SBB log lines with one line every loginterval nodes</td>
</tr>
<tr>
<td>1</td>
<td>NLP solver log for the root node plus SBB loglines as 0</td>
</tr>
<tr>
<td>2</td>
<td>NLP solver log for all nodes plus SBB log lines as 0</td>
</tr>
</tbody>
</table>

memnodes {integer}: maximum number of nodes in memory ➞

The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: 10000

miptrace {string}: filename of MIP trace file ➞

More info is available in chapter Solve trace

miptracenode {integer}: node interval when a trace record is written ➞

More info is available in chapter Solve trace

Default: 100

miptracetime {real}: time interval when a trace record is written ➞

More info is available in chapter Solve trace

Default: 5.0

nodesel {integer}: node selection strategy ➞

Node selection scheme.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>automatic</td>
</tr>
<tr>
<td>1</td>
<td>Depth First Search (DFS)</td>
</tr>
<tr>
<td>2</td>
<td>Best Bound (BB)</td>
</tr>
<tr>
<td>3</td>
<td>Best Estimate (BE)</td>
</tr>
<tr>
<td>4</td>
<td>DFS/BB mix</td>
</tr>
<tr>
<td>5</td>
<td>DFS/BE mix</td>
</tr>
<tr>
<td>6</td>
<td>DFS/BB/BE mix</td>
</tr>
</tbody>
</table>

**printbbinfo (integer):** prints additional node info

Additional info of log output.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>print no additional info</td>
</tr>
<tr>
<td>1</td>
<td>print variable selection letter</td>
</tr>
<tr>
<td></td>
<td>The node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.</td>
</tr>
<tr>
<td>2</td>
<td>print best estimate</td>
</tr>
</tbody>
</table>

**rootsolver (string):** solver for the root node

`solver[,n]` Solver is the name of the GAMS NLP solver that should be used in the root node, and `n` is the integer corresponding to optfile for the root node. If `n` is missing, the optfile treated as zero i.e. the NLP solver will not look for an options file. This SBB option can be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: GAMS NLP solver

**solvelink (integer):** Solvelink for GAMS NLP solver

Default: 5

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Call GAMS NLP solver via script</td>
</tr>
<tr>
<td>2</td>
<td>Call GAMS NLP solver via module</td>
</tr>
<tr>
<td>5</td>
<td>Call GAMS NLP solver in memory</td>
</tr>
</tbody>
</table>

**subiter (integer):** iteration limit for the subsolve

The default for `subiter` passed on through `iterlim`. Similar to `subres` but sets the iteration limit for solving a node in the B&B tree.

Default: GAMS iterlim
subres (real): resource limit for the subsolve

The default for subres passed on through reslim. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.

Default: GAMS reslim

subsolver (string): solver for the subproblems

solver[n] Similar to rootsolver but applied to the subnodes.

Default: GAMS NLP solver

usercallparmfile (string): Command-line parameter include file used in GAMS command-line calls triggered by BCH

usergdxname (string): the name of the GDX file exported from the solver with the solution at the node

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchout.gdx

usergdxnameinc (string): the name of the GDX file exported from the solver with the incumbent solution

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: bchout_i.gdx

userheurcall (string): the GAMS command line to call the heuristic

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userheurfirst (integer): calls the cut generator for the first n nodes

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

userheurfreq (integer): determines the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

userheurinterval (integer): determines the interval when to apply the multiplier for the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 100

userheurmult (integer): determines the multiplier for the frequency of the cut generator model calls

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 2

userheurnewint (boolean): calls the heuristic if the solver found a new integer feasible solution

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 0

userheurobjfirst (integer): Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 50

varsel (integer): variable selection strategy at each node

Variable selection scheme.

Default: 0
5.35.5 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the loginterval and loglevel options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

<table>
<thead>
<tr>
<th>Node</th>
<th>Act. Lev.</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Int.</th>
<th>Best Bound</th>
<th>Gap (2 secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8457.6878</td>
<td>3</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8491.2869</td>
<td>2</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>8518.1779</td>
<td>1</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
<td>9338.1020</td>
<td>0</td>
<td>9338.1020</td>
<td>8457.6878</td>
<td>0.1041</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>pruned</td>
<td>9338.1020</td>
<td>8491.2869</td>
<td>0.0997</td>
</tr>
</tbody>
</table>

Solution satisfies optcr

Statistics:

- Iterations: 90
- NLP Seconds: 0.110000
- B&B nodes: 3
- MIP solution: 9338.101979 found in node 3
- Best possible: 8491.286941
- Absolute gap: 846.815039 optca: 0.000000
- Relative gap: 0.099728 optcr: 0.100000
- Model Status: 8
- Solver Status: 1

NLP Solver Statistics

- Total Number of NLP solves: 7
- Total Number of NLP failures: 0
- Details: conopt
  - # execs: 7
  - # failures: 0

Terminating.

The fields in the log are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>The number of the current node. The root node is node 0.</td>
</tr>
<tr>
<td>Act</td>
<td>The number of active nodes defined as the number of subnodes that have not yet been solved.</td>
</tr>
<tr>
<td>Lev</td>
<td>The level in the search tree, i.e., the number of branches needed to reach this node.</td>
</tr>
<tr>
<td>Objective</td>
<td>The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. &quot;pruned&quot; indicates that the objective value was worse than the best integer value, &quot;infeasible&quot; indicates that the node was Infeasible or Locally Infeasible, and &quot;ignored&quot; indicates that the node could not be solved (see under failseq above).</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IInf</td>
<td>The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to IInf.</td>
</tr>
<tr>
<td>Best Int</td>
<td>The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (*) in column one indicates that the node is integer and that the solution is better than the best yet found.</td>
</tr>
<tr>
<td>Best Bound</td>
<td>The minimum value of &quot;Objective&quot; for the subnodes that have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.</td>
</tr>
<tr>
<td>Gap</td>
<td>The relative gap between the Best Integer solution and the Best Bound.</td>
</tr>
</tbody>
</table>

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text "ignored" in the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

```
Root node solved locally optimal.
Node Act. Lev. Objective IInf Best Int. Best Bound Gap (2 secs)
0   0   0    6046.0186   12   -    6046.0186   -
1   1   1    infeasible    -    -    6046.0186   -
2   0   1    6042.0995   10   -    6042.0995   -
3   1   2    ignored    -    -    6042.0995   - (4,6)
4   0   2    5804.5856   8    -    5804.5856   -
5   1   3    ignored    -    -    5804.5856   - (4,7)
```

The next Log file shows the effect of the `infeasseq` and `failseq` options on the model above. CONOPT with options file `conopt.opt` (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT with option file fails but CONOPT without option file finds a Locally Optimal solution, and this solution is then used for further search. The SBB option file for the following run would be:

```
rootsolver conopt.1
subsolver conopt.1
failseq conopt
infeasseq 100 conopt minos snopt
```

The log looks as follows:

```
Root node solved locally optimal.
Node Act. Lev. Objective IInf Best Int. Best Bound Gap (2 secs)
0   0   0    6046.0186   12   -    6046.0186   -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos reports locally infeasible
```
Executing snopt
1 1 1 infeasible - - 6046.0186 -
2 0 1 6042.0995 10 - 6042.0995 -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 FEASIBLE SOLUTION
Executing conopt
3 1 2 4790.2373 8 - 6042.0995 -
4 2 3 4481.4156 6 - 6042.0995 -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE
Executing snopt
5 3 4 infeasible - - 6042.0995 -
6 2 4 4480.3778 4 - 6042.0995 -

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

NLP Solver Statistics
Total Number of NLP solves : 45
Total Number of NLP failures: 13
Details: conopt minos snopt
# execs 34 3 8
# failures 4 3 6

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These jumps of the objective in the wrong direction which might also have an impact on the best bound/possible are reported in a separate statistic:

Non convex model!
# jumps in best bound : 2
Maximum jump in best bound : 20.626587 in node 13
# jumps to better objective : 2
Maximum jump in objective : 20.626587 in node 13

5.35.6 Comparison of SBB and other MINLP Solvers

GAMS offers a variety of MINLP solvers including local and global MINLP solver. They implement different algorithms and it is usually unclear which solver performs best. Here we give a brief comparison between SBB and the well known solver DICOPT.

DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.
5.36 SCIP

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using much cheaper linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.

SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored.

Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).

5.36 SCIP

SCIP (Solving Constraint Integer Programs) is a framework for Constraint Integer Programming oriented towards the needs of Mathematical Programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP or MINLP solver or as a framework for branch-cut-and-price. Within GAMS, the MIP and MINLP solving facilities of SCIP are available.

For more detailed information, we refer to [4] [1] [2] [21] [20] [74] [127] [88] [89] [75] [22] [192] [191] [204] and the SCIP web site.

GAMS/SCIP uses CPLEX, if licensed, and otherwise SOPLEX [206] as LP solver, the COIN-OR Interior Point Optimizer IPOPT [197] as nonlinear solver, and CppAD to compute derivatives of nonlinear functions.

SCIP supports continuous, binary, integer, semi-continuous, semi-integer variables, indicator constraints, special ordered sets, and branching priorities for discrete variables.

5.36.1 Usage

The following statement can be used inside your GAMS program to specify using SCIP

Option MIP = SCIP; \{ or QCP or NLP or MIQCP or MINLP or ... \}

The above statement should appear before the Solve statement. If SCIP was specified as the default solver during GAMS installation, the above statement is not necessary.
5.36.1.1 Specification of SCIP Options

GAMS/SCIP supports the GAMS parameters `reslim`, `iterlim`, `nodlim`, `optca`, `optcr`, and `workspace`. Further, the option `threads` can be used to control the number of threads used in the MILP presolver, the linear algebra routines of IPOPT, and for solving LPs if CPLEX' barrier solver is used.

Options can be specified by a SCIP options file. A SCIP options file consists of one option or comment per line. A pound sign (`#`) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (`=`) and any amount of white space (blanks or tabs). Further, string values have to be enclosed in quotation marks.

A small example for a scip.opt file is:

```
propagating/probing/maxprerounds = 0
separating/maxrounds = 0
separating/maxroundsroot = 0
```

It causes GAMS/SCIP to disable probing during presolve and to turn off all cut generators.

5.36.1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to SCIP via a separate file, see Indicator Constraints for more details on its syntax. The name of that file is specified via the option `gams/indicatorfile` in a SCIP option file. Currently, indicators can only be used for linear equations.

5.36.2 Special Features

5.36.2.1 SCIP interactive shell

The interactive shell in SCIP is a powerful tool that allows the user to display various information (e.g., branching statistics, presolved model), load emphasis settings, interrupt a solve to change parameters or trigger a restart, write the model in various file formats, start SCIP's solution counter, and many more things.

When setting the option `gams/interactive` to a nonempty string, the GAMS/SCIP interface opens the interactive shell of SCIP after having load the GAMS problem and parameters and passes the value of the `gams/interactive` parameter to the SCIP interactive shell.

By default, SCIP behaves as if `gams/interactive` has been set to "optimize write gamssol quit", that is, SCIP is requested to solve the problem, then to pass the solution back to GAMS, and to quit.

An example use of the SCIP interactive shell feature via GAMS is to add the following line to your SCIP options file:

```
gams/interactive = "write prob orig.lp presolve write transprob presol.mps opt write gamssol quit"
```

This instructs SCIP to write the original problem to the file `orig.lp` in LP format, to presolve the instance, to write the presolved problem to the file `presolved.mps` in MPS format, to solve the instance, to write the solution out to GAMS, and to finish.

By omitting the `quit` command, SCIP's interactive shell remains open and awaits user input. The command `help` prints a list of available commands. Note, that on Windows, GAMS need to be called with the option `interactivesolver` enabled to allow user input for the solver process.

A tutorial on using the SCIP shell is available at https://www.scipopt.org/doc/html/SHELL.php.
5.36.2.2 Emphasis Settings

SCIP includes various emphasis settings, which are predefined values for a set of SCIP parameters. Such predefined settings are available for setting the effort that SCIP should spend for, e.g., presolving, separation, or heuristics.

The emphasis settings are not available as single parameters, but can be set via SCIPs interactive shell. E.g., writing `set heuristics emphasis` in the shell displays the available emphasis settings for heuristics (`aggressive, fast, off`) and expects the user to input which setting to use. Further, general emphasis settings are available in the `set emphasis` menu, some of them giving predefined settings similar to the CPLEX option `mipemphasis`.

Via the `gams/interactive` option, c.f. Section SCIP interactive shell, emphasis settings can be activated via a SCIP options file. For example, adding the option

```
gams/interactive = "set emphasis feasibility set loadgams optimize write gamssol quit"
```

instructs SCIP to load the emphasis setting `feasibility` prior to optimizing the model and passing the solution back to GAMS. Note, that setting one of the emphasis settings `feasibility, hardlp, and optimality` resets all previously set parameters to their default values, which includes the ones that are set by the GAMS/SCIP interface or which were loaded from a SCIP options file. Therefore, the command `set loadgams` has been used above to restore these parameter settings.

The following emphasis settings are available in SCIP:

<table>
<thead>
<tr>
<th>shell command</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>set emphasis easycip</td>
<td>use for easy problems</td>
</tr>
<tr>
<td>set emphasis feasibility</td>
<td>emphasize finding a feasible solution</td>
</tr>
<tr>
<td>set emphasis hardlp</td>
<td>use for problems with a hard LP</td>
</tr>
<tr>
<td>set emphasis optimality</td>
<td>emphasize proving optimality</td>
</tr>
<tr>
<td>set emphasis numerics</td>
<td>increase numerical stability</td>
</tr>
<tr>
<td>set heuristic emphasis</td>
<td>use primal heuristics aggressively</td>
</tr>
<tr>
<td>set presolving emphasis</td>
<td>use only fast presolving steps</td>
</tr>
<tr>
<td>set presolving emphasis</td>
<td>disable all primal heuristics</td>
</tr>
<tr>
<td>set separating emphasis</td>
<td>use cutting plane separators aggressively</td>
</tr>
<tr>
<td>set separate emphasis</td>
<td>use only fast cutting plane separators</td>
</tr>
<tr>
<td>set separate emphasis</td>
<td>disable all cutting plane separators</td>
</tr>
</tbody>
</table>

5.36.2.3 Starting point

Using the `completesol` heuristic, SCIP can try to find a feasible solution based on values given by the user for all or some of the variables. The values need to be specified as variable levels in the GAMS model. The heuristic solves a copy of the problem where variables for which values have been provided are restricted to be close to that value. When an integral value is specified for a binary or integer variable, the variable is fixed to that value. See also Section 2.3.2 in [127] for a more detailed description of the heuristic.

For which variables the level values are passed from GAMS to SCIP is controlled by the parameter `gams/mipstart`. The parameter values have the following meaning:
• 0: do not pass any variable values to SCIP, the heuristic will not run
• 1: pass values for all binary and integer variables to SCIP and let SCIP try to find a feasible solution in its neighborhood by using the completesol heuristic
• 2 (default): pass values for all variables to SCIP and let SCIP check feasibility of the given solution, the heuristic will not run
• 3: pass values for all variables to SCIP and let SCIP try to find a feasible solution in the neighborhood by using the completesol heuristic
• 4: pass values for all binary and integer variables to SCIP which fractionality is at most the value of GAMS option `tryint` (thus, with default `tryint=0`, only for variables with integral values, the value is passed to SCIP) and let SCIP try to find a feasible solution in the neighborhood by using the completesol heuristic

Note, that the completesol heuristic will not run if there are too many variables with unknown values. This behavior can be adjusted by setting parameter `heuristics/completesol/maxunknownrate`.

5.36.2.4 Decomposition Information

SCIP potentially can make use of a user-provided problem decomposition, e.g., in primal heuristics PADM or GINS. Note, that SCIP’s Benders Decomposition is not available with GAMS at the moment. A GAMS user can inform SCIP about such a decomposition by using the .stage variable suffix.

In an attempt to following the same convention on interpreting the .stage variable suffix as the GAMS/CPLEX link, variables which .stage suffix is set to an integral value greater than 0 are assigned to a block with this number. Variables which .stage suffix is set to a fractional value or a value smaller than 1 are assigned to no block.

Note, that the default value for the .stage suffix is 1. Therefore, GAMS/SCIP only passes decomposition information to SCIP if at least one variable has its .stage suffix set to a value different than 1. Moreover, in order to receive the .stage values, `scaleOpt` and `priorOpt` must be left at their default of 0.

5.36.2.5 Solution Pool

When SCIP solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option `gams/dumpsolutions` is specified, then all alternative solutions found by SCIP are written into GDX files and an index file with the name given by the this option is written. If the option `gams/dumpsolutionsmerged` is specified, then all alternative solutions found by SCIP are written into a single GDX file, which name is given by the this option.

The GAMS testlib model `dumpsol` shows an example use for this option.

5.36.2.6 Solving process tracing

The option `gams/solvetrace/file` can be used to specify the name of a file where information about the progress of the branch-and-bound tree search in SCIP is stored. The file is created and updated during the solution process, so it may also be used to monitor the progress of SCIP while it still solves the model.

New entries are written periodically, depending on how many nodes have been processed or how much time has been elapsed since the last entry was written. Each entry contains information on the current primal and dual bound.
5.36.2.7 Notes on solving MINLPs

SCIP includes capabilities to handle nonlinear functions that are specified via algebraic expressions. Therefore, neither external/extrinsic functions nor all GAMS mathematical functions are supported.

Nonconvex MINLPs are solved via a spatial branch-and-bound algorithm using linear relaxations. The tightness of this relaxation depends heavily on the variable bounds, thus tight bounds for the nonlinear variables are crucial for SCIP.

Convex MINLPs are much easier to solve for SCIP, provided it recognizes the convexity of the model. The convexity check implemented in SCIP may not give a conclusive answer in all cases. However, setting the option constraints/nonlinear/assumeconvex to TRUE can be used to tell SCIP that it should assume all nonlinear constraints to be of convex type. This may help to improve solving times for convex MINLPs considerably in some cases. However, it can also deteriorate performance as it prevents the use of certain extended formulations.

Another feature that can be especially useful for convex MINLPs is to enable the generation of cuts in feasible solutions, since these supporting hyperplanes can be strong. At the moment, this is only enabled for purely continuous nonlinear programs (QCPs and NLPs), but can be adjusted by parameter constraints/nonlinear/linearizeheursol. In this connection, it may also be beneficial to set separating/poolfreq to 1.

5.36.3 Components

In the following, we list components that are available in SCIP together with some common properties. Many of these properties can be modified with corresponding parameters.

5.36.3.1 Branching Rules

<table>
<thead>
<tr>
<th>branching rule</th>
<th>priority</th>
<th>maxdepth</th>
<th>maxbounddist</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relpscost</td>
<td>10000</td>
<td>-1</td>
<td>100</td>
<td>reliability branching on pseudo cost values</td>
</tr>
<tr>
<td>pscost</td>
<td>2000</td>
<td>-1</td>
<td>100</td>
<td>branching on pseudo cost values</td>
</tr>
<tr>
<td>inference</td>
<td>1000</td>
<td>-1</td>
<td>100</td>
<td>inference history branching</td>
</tr>
<tr>
<td>mostinf</td>
<td>100</td>
<td>-1</td>
<td>100</td>
<td>most infeasible branching</td>
</tr>
<tr>
<td>leastinf</td>
<td>50</td>
<td>-1</td>
<td>100</td>
<td>least infeasible branching</td>
</tr>
<tr>
<td>distribution</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>branching rule based on variable influence on cumulative normal distribution of row activities</td>
</tr>
<tr>
<td>fullstrong</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>full strong branching</td>
</tr>
<tr>
<td>cloud</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>branching rule that considers several alternative LP optima</td>
</tr>
<tr>
<td>lookahead</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>full strong branching over multiple levels</td>
</tr>
<tr>
<td>multaggr</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>fullstrong branching on fractional and multi-aggregated variables</td>
</tr>
<tr>
<td>allfullstrong</td>
<td>-1000</td>
<td>-1</td>
<td>100</td>
<td>all variables full strong branching</td>
</tr>
<tr>
<td>vanillafullstrong</td>
<td>-2000</td>
<td>-1</td>
<td>100</td>
<td>vanilla full strong branching</td>
</tr>
<tr>
<td>random</td>
<td>-100000</td>
<td>-1</td>
<td>100</td>
<td>random variable branching</td>
</tr>
<tr>
<td>nodereopt</td>
<td>-9000000</td>
<td>-1</td>
<td>100</td>
<td>branching rule for node reoptimization</td>
</tr>
</tbody>
</table>

5.36.3.2 Conflict Handler

<table>
<thead>
<tr>
<th>conflict handler</th>
<th>priority</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logicor</td>
<td>800000</td>
<td>conflict handler creating logic or constraints</td>
</tr>
<tr>
<td>setppc</td>
<td>700000</td>
<td>conflict handler creating set covering constraints</td>
</tr>
<tr>
<td>indicatorconflict</td>
<td>200000</td>
<td>replace slack variables and generate logicor constraints</td>
</tr>
<tr>
<td>linear</td>
<td>-1000000</td>
<td>conflict handler creating linear constraints</td>
</tr>
<tr>
<td>bounddisjunction</td>
<td>-3000000</td>
<td>conflict handler creating bound disjunction constraints</td>
</tr>
</tbody>
</table>

5.36.3.3 Constraint Handler

<table>
<thead>
<tr>
<th>constraint handler</th>
<th>check-prio</th>
<th>enfo-prio</th>
<th>sepa-prio</th>
<th>sepa-freq</th>
<th>propfreq</th>
<th>eager-freq</th>
<th>presolve-veitings</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integral</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>always</td>
<td>integrality constraint</td>
</tr>
<tr>
<td>SOS1</td>
<td>-10</td>
<td>100</td>
<td>1000</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>medium</td>
<td>SOS1 constraint handler</td>
</tr>
<tr>
<td>SOS2</td>
<td>-10</td>
<td>100</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>fast</td>
<td>SOS2 constraint handler</td>
</tr>
<tr>
<td>varbound</td>
<td>-500000</td>
<td>-500000</td>
<td>900000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>fast</td>
<td>variable bounds lhs &lt;= x + c*y &lt;= rhs, x non-binary, y non-continuous</td>
</tr>
<tr>
<td>knapsack</td>
<td>-600000</td>
<td>-600000</td>
<td>600000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>always</td>
<td>knapsack constraint of the form ( a^T x \leq b ), x binary and a \geq 0</td>
</tr>
<tr>
<td>setppc</td>
<td>-700000</td>
<td>-700000</td>
<td>700000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>always</td>
<td>set partitioning / packing / covering constraints</td>
</tr>
<tr>
<td>constraint handler</td>
<td>check-prio</td>
<td>enfo-prio</td>
<td>sepa-prio</td>
<td>sepa-freq</td>
<td>propfreq</td>
<td>eager-freq</td>
<td>presol-vetimings</td>
<td>description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>or</td>
<td>-8500000</td>
<td>-850000</td>
<td>850000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>medium</td>
<td>constraint handler for or constraints: ( r = \text{or}(x_1, ..., x_n) )</td>
</tr>
<tr>
<td>and</td>
<td>-850100</td>
<td>-850100</td>
<td>850100</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>fast exhaustive</td>
<td>constraint handler for AND-constraints: ( r = \text{and}(x_1, ..., x_n) )</td>
</tr>
<tr>
<td>xor</td>
<td>-850200</td>
<td>-850200</td>
<td>850200</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>always</td>
<td>constraint handler for xor constraints: ( r = \text{xor}(x_1, ..., x_n) )</td>
</tr>
<tr>
<td>linear</td>
<td>-1000000</td>
<td>-1000000</td>
<td>100000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>fast exhaustive</td>
<td>linear constraints of the form ( \text{lhs} \leq a^T x \leq \text{rhs} )</td>
</tr>
<tr>
<td>orbisack</td>
<td>-1005200</td>
<td>-1005200</td>
<td>40100</td>
<td>5</td>
<td>5</td>
<td>-1</td>
<td>exhaustive</td>
<td>symmetry breaking constraint handler for orbisacks</td>
</tr>
<tr>
<td>orbitope</td>
<td>-1005200</td>
<td>-1005200</td>
<td>40100</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>medium</td>
<td>symmetry breaking constraint handler relying on (partitioning/packing) orbitopes</td>
</tr>
<tr>
<td>constraint handler</td>
<td>check-prio</td>
<td>enfo-prio</td>
<td>sepa-prio</td>
<td>sepa-freq</td>
<td>propfreq</td>
<td>eager-freq</td>
<td>presol-vetimings</td>
<td>description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>symresack</td>
<td>-1005200</td>
<td>-1005200</td>
<td>40100</td>
<td>5</td>
<td>5</td>
<td>-1</td>
<td>exhaustive</td>
<td>symmetry breaking constraint handler relying on symresacks</td>
</tr>
<tr>
<td>logicor</td>
<td>-2000000</td>
<td>-2000000</td>
<td>10000</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>always</td>
<td>logic or constraints</td>
</tr>
<tr>
<td>bounddisj</td>
<td>-3000000</td>
<td>-3000000</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>100</td>
<td>fast</td>
<td>bound disjunction constraints</td>
</tr>
<tr>
<td>nonlinear</td>
<td>-4000010</td>
<td>-60</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>always</td>
<td>handler for non-linear constraints specified by algebraic expressions</td>
</tr>
<tr>
<td>indicator</td>
<td>-6000000</td>
<td>-100</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>fast</td>
<td>indicator constraint handler</td>
</tr>
<tr>
<td>components</td>
<td>-9999999</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>independent components constraint handler</td>
<td></td>
</tr>
</tbody>
</table>


### 5.36.3.4 Display Columns

<table>
<thead>
<tr>
<th>display column</th>
<th>column header</th>
<th>position</th>
<th>width</th>
<th>priority</th>
<th>status</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solfound</td>
<td></td>
<td>0</td>
<td>1</td>
<td>80000</td>
<td>auto</td>
<td>letter that indicates the heuristic which found the solution</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
<td>50</td>
<td>5</td>
<td>4000</td>
<td>auto</td>
<td>total solution time</td>
</tr>
<tr>
<td>nnodes</td>
<td>node</td>
<td>100</td>
<td>7</td>
<td>100000</td>
<td>auto</td>
<td>number of processed nodes</td>
</tr>
<tr>
<td>display column</td>
<td>header</td>
<td>position</td>
<td>width</td>
<td>priority</td>
<td>status</td>
<td>description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>----------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nodesleft</td>
<td>left</td>
<td>200</td>
<td>7</td>
<td>90000</td>
<td>auto</td>
<td>number of unprocessed nodes</td>
</tr>
<tr>
<td>nrank1nodes</td>
<td>rank1</td>
<td>500</td>
<td>7</td>
<td>40000</td>
<td>off</td>
<td>current number of rank1 nodes left</td>
</tr>
<tr>
<td>nnodesbelowinc</td>
<td>nbInc</td>
<td>550</td>
<td>6</td>
<td>40000</td>
<td>off</td>
<td>current number of nodes with an estimate better than the current incumbent</td>
</tr>
<tr>
<td>lpiterations</td>
<td>LP iter</td>
<td>1000</td>
<td>7</td>
<td>30000</td>
<td>auto</td>
<td>number of simplex iterations</td>
</tr>
<tr>
<td>lpavgiterations</td>
<td>LP it/n</td>
<td>1400</td>
<td>7</td>
<td>25000</td>
<td>auto</td>
<td>average number of LP iterations since the last output line</td>
</tr>
<tr>
<td>lpcond</td>
<td>LP cond</td>
<td>1450</td>
<td>7</td>
<td>0</td>
<td>auto</td>
<td>estimate on condition number of LP solution</td>
</tr>
<tr>
<td>memused</td>
<td>umem</td>
<td>1500</td>
<td>5</td>
<td>0</td>
<td>auto</td>
<td>total number of bytes used in block memory</td>
</tr>
<tr>
<td>memtotal</td>
<td>mem/heur</td>
<td>1500</td>
<td>8</td>
<td>20000</td>
<td>auto</td>
<td>total number of bytes in block memory or the creator name when a new incumbent solution was found</td>
</tr>
<tr>
<td>depth</td>
<td>depth</td>
<td>2000</td>
<td>5</td>
<td>500</td>
<td>auto</td>
<td>depth of current node</td>
</tr>
<tr>
<td>maxdepth</td>
<td>mdpt</td>
<td>2100</td>
<td>5</td>
<td>5000</td>
<td>auto</td>
<td>maximal depth of all processed nodes</td>
</tr>
<tr>
<td>plungedepth</td>
<td>pdpt</td>
<td>2200</td>
<td>5</td>
<td>10</td>
<td>auto</td>
<td>current plunging depth</td>
</tr>
<tr>
<td>nfrac</td>
<td>frac</td>
<td>2500</td>
<td>5</td>
<td>700</td>
<td>auto</td>
<td>number of fractional variables in the current solution</td>
</tr>
<tr>
<td>nexternbranchcands</td>
<td>extbr</td>
<td>2600</td>
<td>5</td>
<td>650</td>
<td>auto</td>
<td>number of extern branching variables in the current node</td>
</tr>
<tr>
<td>vars</td>
<td>vars</td>
<td>3000</td>
<td>5</td>
<td>3000</td>
<td>auto</td>
<td>number of variables in the problem</td>
</tr>
<tr>
<td>conss</td>
<td>cons</td>
<td>3100</td>
<td>5</td>
<td>3100</td>
<td>auto</td>
<td>number of globally valid constraints in the problem</td>
</tr>
<tr>
<td>curconss</td>
<td>ccons</td>
<td>3200</td>
<td>5</td>
<td>600</td>
<td>auto</td>
<td>number of enabled constraints in current node</td>
</tr>
<tr>
<td>display column</td>
<td>header</td>
<td>position</td>
<td>width</td>
<td>priority</td>
<td>status</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>----------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>curcols</td>
<td>cols</td>
<td>3300</td>
<td>5</td>
<td>800</td>
<td>auto</td>
<td>number of LP columns in current node</td>
</tr>
<tr>
<td>currows</td>
<td>rows</td>
<td>3400</td>
<td>5</td>
<td>900</td>
<td>auto</td>
<td>number of LP rows in current node</td>
</tr>
<tr>
<td>cuts</td>
<td>cuts</td>
<td>3500</td>
<td>5</td>
<td>2100</td>
<td>auto</td>
<td>total number of cuts applied to the LPs</td>
</tr>
<tr>
<td>separounds</td>
<td>sepa</td>
<td>3600</td>
<td>4</td>
<td>100</td>
<td>auto</td>
<td>number of separation rounds performed at the current node</td>
</tr>
<tr>
<td>poolsize</td>
<td>pool</td>
<td>3700</td>
<td>5</td>
<td>50</td>
<td>auto</td>
<td>number of LP rows in the cut pool</td>
</tr>
<tr>
<td>conflicts</td>
<td>confs</td>
<td>4000</td>
<td>5</td>
<td>2000</td>
<td>auto</td>
<td>total number of conflicts found in conflict analysis</td>
</tr>
<tr>
<td>strongbranches</td>
<td>strbr</td>
<td>5000</td>
<td>5</td>
<td>1000</td>
<td>auto</td>
<td>total number of strong branching calls</td>
</tr>
<tr>
<td>pseudoobj</td>
<td>pseudoobj</td>
<td>6000</td>
<td>14</td>
<td>300</td>
<td>auto</td>
<td>current pseudo objective value</td>
</tr>
<tr>
<td>lpobj</td>
<td>lpobj</td>
<td>6500</td>
<td>14</td>
<td>300</td>
<td>auto</td>
<td>current LP objective value</td>
</tr>
<tr>
<td>curdualbound</td>
<td>curdualbound</td>
<td>7000</td>
<td>14</td>
<td>400</td>
<td>auto</td>
<td>dual bound of current node</td>
</tr>
<tr>
<td>estimate</td>
<td>estimate</td>
<td>7500</td>
<td>14</td>
<td>200</td>
<td>auto</td>
<td>estimated value of feasible solution in current node</td>
</tr>
<tr>
<td>avgdualbound</td>
<td>avgdualbound</td>
<td>8000</td>
<td>14</td>
<td>40</td>
<td>auto</td>
<td>average dual bound of all unprocessed nodes</td>
</tr>
<tr>
<td>dualbound</td>
<td>dualbound</td>
<td>9000</td>
<td>14</td>
<td>70000</td>
<td>auto</td>
<td>current global dual bound</td>
</tr>
<tr>
<td>primalbound</td>
<td>primalbound</td>
<td>10000</td>
<td>14</td>
<td>80000</td>
<td>auto</td>
<td>current primal bound</td>
</tr>
<tr>
<td>cutoffbound</td>
<td>cutoffbound</td>
<td>10100</td>
<td>14</td>
<td>10</td>
<td>auto</td>
<td>current cutoff bound</td>
</tr>
<tr>
<td>gap</td>
<td>gap</td>
<td>20000</td>
<td>8</td>
<td>60000</td>
<td>auto</td>
<td>current (relative) gap using $</td>
</tr>
<tr>
<td>primalgap</td>
<td>primgap</td>
<td>21000</td>
<td>8</td>
<td>20000</td>
<td>off</td>
<td>current (relative) gap using $</td>
</tr>
<tr>
<td>nsols</td>
<td>nsols</td>
<td>30000</td>
<td>5</td>
<td>0</td>
<td>auto</td>
<td>current number of solutions found</td>
</tr>
</tbody>
</table>
### 5.36.3.5 Nonlinear Handler

<table>
<thead>
<tr>
<th>nonlinear handler</th>
<th>enabled</th>
<th>detect priority</th>
<th>enforce priority</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>default handler for expressions</td>
</tr>
<tr>
<td>convex</td>
<td>1</td>
<td>50</td>
<td>50</td>
<td>handler that identifies and estimates convex expressions</td>
</tr>
<tr>
<td>concave</td>
<td>1</td>
<td>40</td>
<td>40</td>
<td>handler that identifies and estimates concave expressions</td>
</tr>
<tr>
<td>bilinear</td>
<td>1</td>
<td>-10</td>
<td>-10</td>
<td>bilinear handler for expressions</td>
</tr>
<tr>
<td>perspective</td>
<td>1</td>
<td>-20</td>
<td>125</td>
<td>perspective handler for expressions</td>
</tr>
<tr>
<td>quadratic</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>handler for quadratic expressions</td>
</tr>
<tr>
<td>quotient</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>nonlinear handler for quotient expressions</td>
</tr>
<tr>
<td>soc</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>nonlinear handler for second-order cone structures</td>
</tr>
</tbody>
</table>

### 5.36.3.6 Node Selectors

<table>
<thead>
<tr>
<th>node selector</th>
<th>standard priority</th>
<th>memsave priority</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate</td>
<td>200000</td>
<td>100</td>
<td>best estimate search</td>
</tr>
<tr>
<td>node selector</td>
<td>standard priority</td>
<td>memsave priority</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>bfs</td>
<td>100000</td>
<td>0</td>
<td>best first search</td>
</tr>
<tr>
<td>hybridestim</td>
<td>50000</td>
<td>50</td>
<td>hybrid best estimate / best bound search</td>
</tr>
<tr>
<td>restartdfs</td>
<td>10000</td>
<td>50000</td>
<td>depth first search with periodical selection  of the best node</td>
</tr>
<tr>
<td>uct</td>
<td>10</td>
<td>0</td>
<td>node selector which balances exploration and exploitation</td>
</tr>
<tr>
<td>dfs</td>
<td>0</td>
<td>100000</td>
<td>depth first search</td>
</tr>
<tr>
<td>breadthfirst</td>
<td>-10000</td>
<td>-1000000</td>
<td>breadth first search</td>
</tr>
</tbody>
</table>


### 5.36.3.7 Presolvers

<table>
<thead>
<tr>
<th>presolver</th>
<th>priority</th>
<th>timing</th>
<th>maxrounds</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>milp</td>
<td>99999999</td>
<td>medium</td>
<td>-1</td>
<td>MILP specific presolving methods</td>
</tr>
<tr>
<td>trivial</td>
<td>90000000</td>
<td>fast</td>
<td>-1</td>
<td>round fractional bounds on integers, fix variables with equal bounds</td>
</tr>
<tr>
<td>boundshift</td>
<td>79000000</td>
<td>fast</td>
<td>0</td>
<td>converts variables with domain [a,b] to variables with domain [0,b-a]</td>
</tr>
<tr>
<td>inttobinary</td>
<td>70000000</td>
<td>fast</td>
<td>-1</td>
<td>converts integer variables with domain [a,a+1] to binaries</td>
</tr>
<tr>
<td>convertinttobin</td>
<td>60000000</td>
<td>fast</td>
<td>0</td>
<td>converts integer variables to binaries</td>
</tr>
<tr>
<td>gateextraction</td>
<td>10000000</td>
<td>exhaustive</td>
<td>-1</td>
<td>presolver extracting gate(and)-constraints</td>
</tr>
<tr>
<td>qpkktref</td>
<td>-1</td>
<td>medium</td>
<td>0</td>
<td>adds KKT conditions to (mixed-binary) quadratic programs</td>
</tr>
<tr>
<td>dualcomp</td>
<td>-50</td>
<td>exhaustive</td>
<td>-1</td>
<td>compensate single up-/downlocks by singleton continuous variables</td>
</tr>
<tr>
<td>stuffing</td>
<td>-100</td>
<td>exhaustive</td>
<td>0</td>
<td>fix redundant singleton continuous variables</td>
</tr>
<tr>
<td>domcol</td>
<td>-1000</td>
<td>exhaustive</td>
<td>-1</td>
<td>dominated column presolver</td>
</tr>
<tr>
<td>tworowbnd</td>
<td>-2000</td>
<td>exhaustive</td>
<td>0</td>
<td>do bound tightening by using two rows</td>
</tr>
<tr>
<td>dualinfe</td>
<td>-3000</td>
<td>exhaustive</td>
<td>0</td>
<td>exploit dual information for fixings and side changes</td>
</tr>
<tr>
<td>implics</td>
<td>-10000</td>
<td>medium</td>
<td>-1</td>
<td>implication graph aggregator</td>
</tr>
<tr>
<td>dualagg</td>
<td>-12000</td>
<td>exhaustive</td>
<td>0</td>
<td>aggregate variables by dual arguments</td>
</tr>
<tr>
<td>sparsify</td>
<td>-24000</td>
<td>exhaustive</td>
<td>-1</td>
<td>eliminate non-zero coefficients</td>
</tr>
<tr>
<td>dualsparsify</td>
<td>-2400000</td>
<td>exhaustive</td>
<td>-1</td>
<td>eliminate non-zero coefficients</td>
</tr>
<tr>
<td>redvub</td>
<td>-9000000</td>
<td>exhaustive</td>
<td>0</td>
<td>detect redundant variable bound constraints</td>
</tr>
</tbody>
</table>


### 5.36.3.8 Primal Heuristics
<table>
<thead>
<tr>
<th>primal heuristic</th>
<th>type</th>
<th>priority</th>
<th>freq</th>
<th>freqoffset</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actconsdiving</td>
<td>d</td>
<td>-1003700</td>
<td>-1</td>
<td>5</td>
<td>LP diving heuristic that chooses fixings w.r.t. the active constraints</td>
</tr>
<tr>
<td>adaptivediving</td>
<td>d</td>
<td>-70000</td>
<td>5</td>
<td>3</td>
<td>diving heuristic that selects adaptively between the existing, public divesets</td>
</tr>
<tr>
<td>bound</td>
<td>p</td>
<td>-1107000</td>
<td>-1</td>
<td>0</td>
<td>heuristic which fixes all integer variables to a bound and solves the remaining LP</td>
</tr>
<tr>
<td>clique</td>
<td>p</td>
<td>5000</td>
<td>0</td>
<td>0</td>
<td>LNS heuristic using a clique partition to restrict the search neighborhood</td>
</tr>
<tr>
<td>coefdiving</td>
<td>d</td>
<td>-1001000</td>
<td>-1</td>
<td>1</td>
<td>LP diving heuristic that chooses fixings w.r.t. the matrix coefficients</td>
</tr>
<tr>
<td>completesol</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>primal heuristic trying to complete given partial solutions</td>
</tr>
<tr>
<td>conflictdiving</td>
<td>d</td>
<td>-1000100</td>
<td>10</td>
<td>0</td>
<td>LP diving heuristic that chooses fixings w.r.t. conflict locks</td>
</tr>
<tr>
<td>crossover</td>
<td>L</td>
<td>-1104000</td>
<td>30</td>
<td>0</td>
<td>LNS heuristic that fixes all variables that are identic in a couple of solutions</td>
</tr>
<tr>
<td>dins</td>
<td>L</td>
<td>-1105000</td>
<td>-1</td>
<td>0</td>
<td>distance induced neighborhood search by Ghosh</td>
</tr>
<tr>
<td>distributiondiving</td>
<td>d</td>
<td>-1003300</td>
<td>10</td>
<td>3</td>
<td>Diving heuristic that chooses fixings w.r.t. changes in the solution density</td>
</tr>
<tr>
<td>dps</td>
<td>L</td>
<td>7500</td>
<td>-1</td>
<td>0</td>
<td>primal heuristic for decomposable MIPs</td>
</tr>
<tr>
<td>dualval</td>
<td>L</td>
<td>-10</td>
<td>-1</td>
<td>0</td>
<td>primal heuristic using dual values</td>
</tr>
<tr>
<td>farkasdiving</td>
<td>d</td>
<td>-900000</td>
<td>10</td>
<td>0</td>
<td>LP diving heuristic that tries to construct a Farkas-proof</td>
</tr>
<tr>
<td>feaspump</td>
<td>o</td>
<td>-1000000</td>
<td>20</td>
<td>0</td>
<td>objective feasibility pump 2.0</td>
</tr>
<tr>
<td>fixandinfer</td>
<td>p</td>
<td>-500000</td>
<td>-1</td>
<td>0</td>
<td>iteratively fixes variables and propagates inferences</td>
</tr>
<tr>
<td>fracdiving</td>
<td>d</td>
<td>-1003000</td>
<td>10</td>
<td>3</td>
<td>LP diving heuristic that chooses fixings w.r.t. the fractionalities</td>
</tr>
<tr>
<td>gins</td>
<td>L</td>
<td>-1103000</td>
<td>20</td>
<td>8</td>
<td>gins works on k-neighborhood in a variable-constraint graph</td>
</tr>
<tr>
<td>guideddiving</td>
<td>d</td>
<td>-1007000</td>
<td>10</td>
<td>7</td>
<td>LP diving heuristic that chooses fixings in direction of incumbent solutions</td>
</tr>
<tr>
<td>zeroobj</td>
<td>L</td>
<td>100</td>
<td>-1</td>
<td>0</td>
<td>heuristic trying to solve the problem without objective</td>
</tr>
<tr>
<td>indicator</td>
<td>L</td>
<td>-20200</td>
<td>1</td>
<td>0</td>
<td>indicator heuristic to create feasible solutions from values for indicator variables</td>
</tr>
<tr>
<td>intrdiving</td>
<td>d</td>
<td>-1003500</td>
<td>-1</td>
<td>9</td>
<td>LP diving heuristic that fixes binary variables with large LP value to one</td>
</tr>
<tr>
<td>intshifting</td>
<td>r</td>
<td>-10000</td>
<td>10</td>
<td>0</td>
<td>LP rounding heuristic with infeasibility recovering and final LP solving</td>
</tr>
<tr>
<td>linesearchdiving</td>
<td>d</td>
<td>-1006000</td>
<td>10</td>
<td>6</td>
<td>LP diving heuristic that chooses fixings following the line from root solution to current solution</td>
</tr>
<tr>
<td>localbranching</td>
<td>L</td>
<td>-1102000</td>
<td>-1</td>
<td>0</td>
<td>local branching heuristic by Fischetti and Lodi</td>
</tr>
<tr>
<td>primal heuristic</td>
<td>type</td>
<td>priority</td>
<td>freq</td>
<td>freqoffset</td>
<td>description</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>locks</td>
<td>p</td>
<td>3000</td>
<td>0</td>
<td>0</td>
<td>heuristic that fixes variables based on their rounding locks</td>
</tr>
<tr>
<td>lface</td>
<td>L</td>
<td>-1104010</td>
<td>15</td>
<td>0</td>
<td>LNS heuristic that searches the optimal LP face inside a sub-MIP</td>
</tr>
<tr>
<td>alns</td>
<td>L</td>
<td>-1100500</td>
<td>20</td>
<td>0</td>
<td>Large neighborhood search heuristic that orchestrates the popular neighborhoods Local Branching, RINS, RENS, DINS etc.</td>
</tr>
<tr>
<td>nlpdiving</td>
<td>d</td>
<td>-1003010</td>
<td>10</td>
<td>3</td>
<td>NLP diving heuristic that chooses fixings w.r.t. the fractionalities</td>
</tr>
<tr>
<td>mutation</td>
<td>L</td>
<td>-1103010</td>
<td>-1</td>
<td>8</td>
<td>mutation heuristic randomly fixing variables</td>
</tr>
<tr>
<td>multistart</td>
<td>L</td>
<td>-2100000</td>
<td>0</td>
<td>0</td>
<td>multistart heuristic randomly fixing variables</td>
</tr>
<tr>
<td>mpec</td>
<td>d</td>
<td>-2050000</td>
<td>50</td>
<td>0</td>
<td>regularization heuristic for convex and nonconvex MINLPs</td>
</tr>
<tr>
<td>objpseudocostdiving</td>
<td>o</td>
<td>-1004000</td>
<td>20</td>
<td>4</td>
<td>LP diving heuristic that changes variable's objective values instead of bounds, using pseudo costs as guide</td>
</tr>
<tr>
<td>octane</td>
<td>r</td>
<td>-1008000</td>
<td>-1</td>
<td>0</td>
<td>octane primal heuristic for pure {0;1}-problems based on Balas et al.</td>
</tr>
<tr>
<td>ofins</td>
<td>L</td>
<td>60000</td>
<td>0</td>
<td>0</td>
<td>primal heuristic for reoptimization, objective function induced neighborhood search</td>
</tr>
<tr>
<td>oneopt</td>
<td>i</td>
<td>-20000</td>
<td>1</td>
<td>0</td>
<td>1-opt heuristic which tries to improve setting of single integer variables</td>
</tr>
<tr>
<td>padm</td>
<td>L</td>
<td>70000</td>
<td>0</td>
<td>0</td>
<td>penalty alternating direction method primal heuristic</td>
</tr>
<tr>
<td>proximity</td>
<td>L</td>
<td>-2000000</td>
<td>-1</td>
<td>0</td>
<td>heuristic trying to improve the incumbent by an auxiliary proximity objective function</td>
</tr>
<tr>
<td>pscostdiving</td>
<td>d</td>
<td>-1002000</td>
<td>10</td>
<td>2</td>
<td>LP diving heuristic that chooses fixings w.r.t. the pseudo cost values</td>
</tr>
<tr>
<td>randrounding</td>
<td>r</td>
<td>-200</td>
<td>20</td>
<td>0</td>
<td>fast LP rounding heuristic</td>
</tr>
<tr>
<td>rens</td>
<td>L</td>
<td>-1100000</td>
<td>0</td>
<td>0</td>
<td>LNS exploring fractional neighborhood of relaxation’s optimum</td>
</tr>
<tr>
<td>reoptsol</td>
<td>p</td>
<td>40000</td>
<td>0</td>
<td>0</td>
<td>primal heuristic updating solutions found in a previous optimization round</td>
</tr>
<tr>
<td>repair</td>
<td>L</td>
<td>-20</td>
<td>-1</td>
<td>0</td>
<td>tries to repair a primal infeasible solution</td>
</tr>
<tr>
<td>rins</td>
<td>L</td>
<td>-1101000</td>
<td>25</td>
<td>0</td>
<td>relaxation induced neighborhood search by Danna, Rothberg, and Le Pape</td>
</tr>
<tr>
<td>rootsoldiving</td>
<td>o</td>
<td>-1005000</td>
<td>20</td>
<td>5</td>
<td>LP diving heuristic that changes variable's objective values using root LP solution as guide</td>
</tr>
<tr>
<td>rounding</td>
<td>r</td>
<td>-100</td>
<td>1</td>
<td>0</td>
<td>LP rounding heuristic with infeasibility recovering</td>
</tr>
<tr>
<td>shiftandpropagate</td>
<td>p</td>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>Pre-root heuristic to expand an auxiliary branch-and-bound tree and apply propagation techniques</td>
</tr>
<tr>
<td>primal heuristic</td>
<td>type</td>
<td>priority</td>
<td>freq</td>
<td>freqoffset</td>
<td>description</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>shifting</td>
<td>r</td>
<td>-5000</td>
<td>10</td>
<td>0</td>
<td>LP rounding heuristic with infeasibility recovering also using continuous variables</td>
</tr>
<tr>
<td>simplerounding</td>
<td>r</td>
<td>-30</td>
<td>1</td>
<td>0</td>
<td>simple and fast LP rounding heuristic</td>
</tr>
<tr>
<td>subnlp</td>
<td>L</td>
<td>-2000010</td>
<td>1</td>
<td>0</td>
<td>primal heuristic that performs a local search in an NLP after fixing integer variables and presolving</td>
</tr>
<tr>
<td>trivial</td>
<td>t</td>
<td>10000</td>
<td>0</td>
<td>0</td>
<td>start heuristic which tries some trivial solutions</td>
</tr>
<tr>
<td>trivialnegation</td>
<td>p</td>
<td>39990</td>
<td>0</td>
<td>0</td>
<td>negate solution entries if an objective coefficient changes the sign, enters or leaves the objective.</td>
</tr>
<tr>
<td>trustregion</td>
<td>L</td>
<td>-1102010</td>
<td>-1</td>
<td>0</td>
<td>LNS heuristic for Benders’ decomposition based on trust region methods</td>
</tr>
<tr>
<td>trysol</td>
<td>t</td>
<td>-3000010</td>
<td>1</td>
<td>0</td>
<td>try solution heuristic</td>
</tr>
<tr>
<td>twoopt</td>
<td>i</td>
<td>-20100</td>
<td>-1</td>
<td>0</td>
<td>primal heuristic to improve incumbent solution by flipping pairs of variables</td>
</tr>
<tr>
<td>undercover</td>
<td>L</td>
<td>-1110000</td>
<td>0</td>
<td>0</td>
<td>solves a sub-CIP determined by a set covering approach</td>
</tr>
<tr>
<td>vbounds</td>
<td>p</td>
<td>2500</td>
<td>0</td>
<td>0</td>
<td>LNS heuristic uses the variable lower and upper bounds to determine the search neighborhood</td>
</tr>
<tr>
<td>veclendiving</td>
<td>d</td>
<td>-1003100</td>
<td>10</td>
<td>4</td>
<td>LP diving heuristic that rounds variables with long column vectors</td>
</tr>
<tr>
<td>zirounding</td>
<td>r</td>
<td>-500</td>
<td>1</td>
<td>0</td>
<td>LP rounding heuristic as suggested by C. Wallace taking row slacks and bounds into account</td>
</tr>
</tbody>
</table>


### 5.36.3.9 Propagators

<table>
<thead>
<tr>
<th>propagator</th>
<th>propprio</th>
<th>freq</th>
<th>presolveprio</th>
<th>presolvetiming</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootredcost</td>
<td>10000000</td>
<td>1</td>
<td>0</td>
<td>always</td>
<td>reduced cost strengthening using root node reduced costs and the cutoff bound</td>
</tr>
<tr>
<td>dualfix</td>
<td>8000000</td>
<td>0</td>
<td>8000000</td>
<td>fast</td>
<td>roundable variables dual fixing</td>
</tr>
<tr>
<td>genvbounds</td>
<td>3000000</td>
<td>1</td>
<td>-2000000</td>
<td>fast</td>
<td>generalized variable bounds propagator</td>
</tr>
<tr>
<td>vbounds</td>
<td>3000000</td>
<td>1</td>
<td>-90000</td>
<td>medium exhaustive</td>
<td>propagates variable upper and lower bounds</td>
</tr>
<tr>
<td>pseudoobj</td>
<td>3000000</td>
<td>1</td>
<td>6000000</td>
<td>fast</td>
<td>pseudo objective function propagator</td>
</tr>
<tr>
<td>redcost</td>
<td>1000000</td>
<td>1</td>
<td>0</td>
<td>always</td>
<td>reduced cost strengthening propagator</td>
</tr>
</tbody>
</table>
### 5.36.3.10 Separators

<table>
<thead>
<tr>
<th>separator</th>
<th>priority</th>
<th>freq</th>
<th>bounddist</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>closecuts</td>
<td>10000000</td>
<td>-1</td>
<td>1</td>
<td>closecuts meta separator</td>
</tr>
<tr>
<td>rlt</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>reformulation-linearization-technique separator</td>
</tr>
<tr>
<td>disjunctive</td>
<td>10d</td>
<td>0</td>
<td>0</td>
<td>disjunctive cut separator</td>
</tr>
<tr>
<td>gauge</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>gauge separator</td>
</tr>
<tr>
<td>interminor</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>intersection cuts separator to ensure that 2x2 minors of X (= xx') have determinant 0</td>
</tr>
<tr>
<td>convexproj</td>
<td>0d</td>
<td>-1</td>
<td>1</td>
<td>separate at projection of point onto convex region</td>
</tr>
<tr>
<td>minor</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>separator to ensure that 2x2 principal minors of X - xx' are positive semi-definite</td>
</tr>
<tr>
<td>impliedbounds</td>
<td>-50</td>
<td>10</td>
<td>1</td>
<td>implied bounds separator</td>
</tr>
<tr>
<td>mixing</td>
<td>-50</td>
<td>10</td>
<td>1</td>
<td>mixing inequality separator</td>
</tr>
<tr>
<td>intobj</td>
<td>-100</td>
<td>-1</td>
<td>0</td>
<td>integer objective value separator</td>
</tr>
<tr>
<td>gomory</td>
<td>-1000</td>
<td>10</td>
<td>1</td>
<td>separator for Gomory mixed-integer and strong CG cuts from LP tableau rows</td>
</tr>
<tr>
<td>cgminip</td>
<td>-1000</td>
<td>-1</td>
<td>0</td>
<td>Chvatal-Gomory cuts via MIPs separator</td>
</tr>
<tr>
<td>aggregation</td>
<td>-3000</td>
<td>10</td>
<td>1</td>
<td>aggregation heuristic for complemented mixed integer rounding cuts and flowcover cuts</td>
</tr>
<tr>
<td>clique</td>
<td>-5000</td>
<td>0</td>
<td>0</td>
<td>clique separator of stable set relaxation</td>
</tr>
<tr>
<td>zerohalf</td>
<td>-6000</td>
<td>10</td>
<td>1</td>
<td>({0,1/2})-cuts separator</td>
</tr>
<tr>
<td>mcf</td>
<td>-10000</td>
<td>0</td>
<td>0</td>
<td>multi-commodity-flow network cut separator</td>
</tr>
<tr>
<td>eccuts</td>
<td>-13000</td>
<td>-1</td>
<td>1</td>
<td>separator for edge-concave functions</td>
</tr>
<tr>
<td>oddcycle</td>
<td>-15000</td>
<td>-1</td>
<td>1</td>
<td>odd cycle separator</td>
</tr>
<tr>
<td>strongcg</td>
<td>-100000</td>
<td>10</td>
<td>0</td>
<td>separator for strong CG cuts</td>
</tr>
<tr>
<td>gomorymi</td>
<td>-100000</td>
<td>10</td>
<td>0</td>
<td>separator for Gomory mixed-integer cuts</td>
</tr>
<tr>
<td>knapsackcover</td>
<td>-100000</td>
<td>10</td>
<td>0</td>
<td>separator for knapsack cover cuts</td>
</tr>
<tr>
<td>flowcover</td>
<td>-100000</td>
<td>10</td>
<td>0</td>
<td>separator for flowcover cuts</td>
</tr>
<tr>
<td>cmir</td>
<td>-100000</td>
<td>10</td>
<td>0</td>
<td>separator for cmir cuts</td>
</tr>
<tr>
<td>rapidlearning</td>
<td>-1200000</td>
<td>5</td>
<td>1</td>
<td>rapid learning heuristic and separator</td>
</tr>
</tbody>
</table>

5.36.4 List of SCIP Options

SCIP supports a large set of options. In the following, we give a detailed list of all SCIP options.

5.36.4.1 gams

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gams/dumpsolutions</td>
<td>name of solutions index gdx file for writing all alternate solutions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>gams/dumpsolutionsmerged</td>
<td>name of gdx file for writing all alternate solutions into a single file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>gams/indicatorfile</td>
<td>name of GAMS options file that contains definitions on indicators</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>gams/interactive</td>
<td>command to be issued to the SCIP shell instead of issuing a solve command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>gams/mipstart</td>
<td>how to handle initial variable levels, see also section Starting point</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 4}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.2 gams/solvtrace

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gams/solvtrace/file</td>
<td>name of file where to write branch-and-bound trace information too</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>gams/solvtrace/nodefreq</td>
<td>frequency in number of nodes when to write branch-and-bound trace information, 0 to disable</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>gams/solvtrace/timefreq</td>
<td>frequency in seconds when to write branch-and-bound trace information, 0.0 to disable</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.3 branching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/clamp</td>
<td>minimal relative distance of branching point to bounds when branching on a continuous variable</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 0.5]</td>
<td></td>
</tr>
<tr>
<td>branching/delaypscostupdate</td>
<td>should updating pseudo costs for continuous variables be delayed to the time after separation?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/divingpscost</td>
<td>should pseudo costs be updated also in diving and probing mode?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/lpgainnormalize</td>
<td>strategy for normalization of LP gain when updating pseudocosts of continuous variables (divide by movement of (l^p) value, reduction in (d')omain width, or reduction in (s')ibling) range: (d, l, s)</td>
<td>s</td>
</tr>
<tr>
<td>branching/midpull</td>
<td>fraction by which to move branching point of a continuous variable towards the middle of the domain; a value of 1.0 leads to branching always in the middle of the domain range: [0, 1]</td>
<td>0.75</td>
</tr>
<tr>
<td>branching/midpullreldomtrig</td>
<td>multiply midpull by relative domain width if the latter is below this value range: [0, 1]</td>
<td>0.5</td>
</tr>
<tr>
<td>branching/preferbinary</td>
<td>should branching on binary variables be preferred? range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>branching/checksol</td>
<td>should LP solutions during strong branching with propagation be checked for feasibility? range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>branching/firstsbchild</td>
<td>child node to be regarded first during strong branching (only with propagation): 'u'p child, 'd'own child, 'h'istory-based, or 'a'utomatic range: a, d, u, h</td>
<td>a</td>
</tr>
<tr>
<td>branching/forceallchildren</td>
<td>should all strong branching children be regarded even if one is detected to be infeasible? (only with propagation) range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>branching/roundsbsol</td>
<td>should LP solutions during strong branching with propagation be rounded? (only when checksbsol=True) range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>branching/scorefac</td>
<td>branching score factor to weigh downward and upward gain prediction in sum score function range: [0, 1]</td>
<td>0.167</td>
</tr>
<tr>
<td>branching/scorefunc</td>
<td>branching score function (‘s’um, ‘p’roduct, ‘q’uotient) range: s, p, q</td>
<td>p</td>
</tr>
<tr>
<td>branching/sumadjustscore</td>
<td>score adjustment near zero by adding epsilon (TRUE) or using maximum (FALSE) range: boolean</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.36.4.5 branching/cloud

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/cloud/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/cloud/maxdepth</td>
<td>maximal depth level, up to which branching rule <code>&lt;cloud&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/cloud/maxdepthunion</td>
<td>maximum depth for the union</td>
<td>65000</td>
</tr>
<tr>
<td>branching/cloud/maxpoints</td>
<td>maximum number of points for the cloud (-1 means no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/cloud/insuccessrate</td>
<td>minimum success rate for the cloud</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/insuccessunion</td>
<td>minimum success rate for the union</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/onlyF2</td>
<td>should only F2 be used?</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/priority</td>
<td>priority of branching rule <code>&lt;cloud&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/usecloud</td>
<td>should a cloud of points be used?</td>
<td>1</td>
</tr>
<tr>
<td>branching/cloud/useunion</td>
<td>should the union of candidates be used?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.6 branching/distribution

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/distribution/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/distribution/maxdepth</td>
<td>maximal depth level, up to which branching rule <code>&lt;distribution&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/distribution/priority</td>
<td>priority of branching rule <code>&lt;distribution&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>branching/distribution/onlyactivecolumns</td>
<td>should only rows which are active at the current node be considered?</td>
<td>0</td>
</tr>
<tr>
<td>branching/distribution/scoreparam</td>
<td>the score;largest 'd'ifference, 'l'owest cumulative probability,'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p.('w')</td>
<td>v</td>
</tr>
</tbody>
</table>
### 5.36.4.7 branching/fullstrong

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/fullstrong/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/fullstrong/maxdepth</td>
<td>maximal depth level, up to which branching rule <code>&lt;fullstrong&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/fullstrong/priority</td>
<td>priority of branching rule <code>&lt;fullstrong&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>branching/fullstrong/forcestrongbranch</td>
<td>should strong branching be applied even if there is just a single candidate?</td>
<td>0</td>
</tr>
<tr>
<td>branching/fullstrong/maxproprounds</td>
<td>maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings)</td>
<td>-2</td>
</tr>
<tr>
<td>branching/fullstrong/probingbounds</td>
<td>should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)?</td>
<td>1</td>
</tr>
<tr>
<td>branching/fullstrong/reevalage</td>
<td>number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.36.4.8 branching/inference

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/inference/conflictprio</td>
<td>priority value for using conflict weights in lex. order</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/cutoffprio</td>
<td>priority value for using cutoff weights in lex. order</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
</tbody>
</table>
### Option Description Default

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>branching/inference/maxdepth</strong></td>
<td>maximal depth level, up to which branching rule <code>&lt;inference&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><strong>branching/inference/priority</strong></td>
<td>priority of branching rule <code>&lt;inference&gt;</code></td>
<td>1000</td>
</tr>
<tr>
<td><strong>branching/inference/useweightedsum</strong></td>
<td>should a weighted sum of inference, conflict and cutoff weights be used?</td>
<td>1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>branching/inference/conflictweight</strong></td>
<td>weight in score calculations for conflict score</td>
<td>1000</td>
</tr>
<tr>
<td><strong>branching/inference/cutoffweight</strong></td>
<td>weight in score calculations for cutoff score</td>
<td>1</td>
</tr>
<tr>
<td><strong>branching/inference/fractionals</strong></td>
<td>should branching on LP solution be restricted to the fractional variables?</td>
<td>1</td>
</tr>
<tr>
<td><strong>branching/inference/inferenceweight</strong></td>
<td>weight in score calculations for inference score</td>
<td>1</td>
</tr>
<tr>
<td><strong>branching/inference/reliablescore</strong></td>
<td>weight in score calculations for conflict score</td>
<td>0.001</td>
</tr>
</tbody>
</table>

#### 5.36.4.9 branching/leastinf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>branching/leastinf/maxbounddist</strong></td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td><strong>branching/leastinf/maxdepth</strong></td>
<td>maximal depth level, up to which branching rule <code>&lt;leastinf&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><strong>branching/leastinf/priority</strong></td>
<td>priority of branching rule <code>&lt;leastinf&gt;</code></td>
<td>50</td>
</tr>
</tbody>
</table>

#### 5.36.4.10 branching/lookahead

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>branching/lookahead/maxbounddist</strong></td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td><strong>branching/lookahead/maxdepth</strong></td>
<td>maximal depth level, up to which branching rule <code>&lt;lookahead&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><strong>branching/lookahead/priority</strong></td>
<td>priority of branching rule <code>&lt;lookahead&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching/lookahead/abbreviated</td>
<td>toggles the abbreviated LAB.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/abbrevpseudo</td>
<td>if abbreviated: Use pseudo costs to estimate the score of a candidate.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/addbinconsrow</td>
<td>should binary constraints be added as rows to the base LP? (0: no, 1: separate, 2: as initial rows)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/addclique</td>
<td>add binary constraints with two variables found at the root node also as a clique</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/addnonviocons</td>
<td>should binary constraints, that are not violated by the base LP, be collected and added?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/applychildbounds</td>
<td>should bounds known for child nodes be applied?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/deeperscoringfunction</td>
<td>scoring function to be used at deeper levels</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Range: d, f, s, w, l, c, r, x</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/enforcemaxdomreds</td>
<td>should the maximum number of domain reductions maxviolateddomreds be enforced?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/filterbymaxgain</td>
<td>should lookahead branching only be applied if the max gain in level 1 is not uniquely that of the best candidate?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/level2avgscore</td>
<td>should the average score be used for uninitialized scores in level 2?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/level2zeroscore</td>
<td>should uninitialized scores in level 2 be set to 0?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/maxncands</td>
<td>if abbreviated: The max number of candidates to consider at the node.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/maxndeepercands</td>
<td>if abbreviated: The max number of candidates to consider per deeper node.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/maxviolatedbincons</td>
<td>how many binary constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/maxviolatedcons</td>
<td>how many constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching/lookahead/maxviolateddomreds</td>
<td>how many domain reductions that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]</td>
<td>1</td>
</tr>
<tr>
<td>branching/lookahead/maxproprounds</td>
<td>maximum number of propagation rounds to perform at each temporary node (-1: unlimited, 0: SCIP default)</td>
<td>0</td>
</tr>
<tr>
<td>branching/lookahead/mergedomainreductions</td>
<td>should domain reductions of feasible siblings should be merged?</td>
<td>0</td>
</tr>
<tr>
<td>branching/lookahead/minweight</td>
<td>if scoringfunction is 's', this value is used to weight the min of the gains of two child problems in the convex combination</td>
<td>0.8</td>
</tr>
<tr>
<td>branching/lookahead/onlyviol domreds</td>
<td>should only domain reductions that violate the LP solution be applied?</td>
<td>0</td>
</tr>
<tr>
<td>branching/lookahead/prefersimplebounds</td>
<td>should domain reductions only be applied if there are simple bound changes?</td>
<td>0</td>
</tr>
<tr>
<td>branching/lookahead/propagate</td>
<td>should domain propagation be executed before each temporary node is solved?</td>
<td>1</td>
</tr>
<tr>
<td>branching/lookahead/recursiondepth</td>
<td>the max depth of LAB.</td>
<td>2</td>
</tr>
<tr>
<td>branching/lookahead/reevalage</td>
<td>max number of LPs solved after which a previous prob branching results are recalculated</td>
<td>10</td>
</tr>
<tr>
<td>branching/lookahead/reevalagefsb</td>
<td>max number of LPs solved after which a previous FSB scoring results are recalculated</td>
<td>10</td>
</tr>
<tr>
<td>branching/lookahead/reusebasis</td>
<td>if abbreviated: Should the information gathered to obtain the best candidates be reused?</td>
<td>1</td>
</tr>
<tr>
<td>branching/lookahead/scoringfunction</td>
<td>scoring function to be used at the base level</td>
<td>a</td>
</tr>
<tr>
<td>branching/lookahead/scoringscoringfunction</td>
<td>scoring function to be used during FSB scoring</td>
<td>d</td>
</tr>
<tr>
<td>branching/lookahead/storeunviolatedsol</td>
<td>if only non violating constraints are added, should the branching decision be stored till the next call?</td>
<td>1</td>
</tr>
<tr>
<td>branching/lookahead/updatebranchingresults</td>
<td>should branching results (and scores) be updated w.r.t. proven dual bounds?</td>
<td>0</td>
</tr>
<tr>
<td>branching/lookahead/usedomainreduction</td>
<td>should domain reductions be collected and applied?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching/lookahead/useimpliedbincons</td>
<td>should binary constraints be collected and applied?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/uselevel2data</td>
<td>should branching data generated at depth level 2 be stored for re-using it?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/lookahead/worsefactor</td>
<td>if the FSB score is of a candidate is worse than the best by this factor, skip this candidate (-1: disable)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.11 branching/mostinf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/mostinf/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>branching/mostinf/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;mostinf&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>branching/mostinf/priority</td>
<td>priority of branching rule &lt;mostinf&gt;</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.12 branching/multaggr

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/multaggr/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>branching/multaggr/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;multaggr&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>branching/multaggr/priority</td>
<td>priority of branching rule &lt;multaggr&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/multaggr/maxproprounds</td>
<td>maximum number of propagation rounds to be performed during multaggr branching before solving the LP (-1: no limit, -2: parameter settings)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-2, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/multaggr/probingbounds</td>
<td>should valid bounds be identified in a probing-like fashion during multaggr branching (only with propagation)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>branching/multaggr/reevalage</td>
<td>number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.13 branching/nodereopt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/nodereopt/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/nodereopt/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;nodereopt&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/nodereopt/priority</td>
<td>priority of branching rule &lt;nodereopt&gt;</td>
<td>-900000</td>
</tr>
</tbody>
</table>

### 5.36.4.14 branching/pscost

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/pscost/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/pscost/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;pscost&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/pscost/narymaxdepth</td>
<td>maximal depth where to do n-ary branching, -1 to turn off</td>
<td>-1</td>
</tr>
<tr>
<td>branching/pscost/naryminwidth</td>
<td>minimal domain width in children when doing n-ary branching, relative to global bounds</td>
<td>0.001</td>
</tr>
<tr>
<td>branching/pscost/narywidthfactor</td>
<td>factor of domain width in n-ary branching when creating nodes with increasing distance from branching value</td>
<td>2</td>
</tr>
<tr>
<td>branching/pscost/nchildren</td>
<td>number of children to create in n-ary branching</td>
<td>2</td>
</tr>
<tr>
<td>branching/pscost/priority</td>
<td>priority of branching rule &lt;pscost&gt;</td>
<td>2000</td>
</tr>
<tr>
<td>branching/pscost/strategy</td>
<td>strategy for utilizing pseudo-costs of external branching candidates (multiply as in pseudo costs 'u'pdate rule, or by 'd'omain reduction, or by domain reduction of 's'ibling, or by 'v'ariable score)</td>
<td>u</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>branching/pscost/maxscoreweight</td>
<td>weight for maximum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>1.3</td>
</tr>
<tr>
<td>branching/pscost/minscoreweight</td>
<td>weight for minimum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>0.8</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching/pscost/sumscoreweight</td>
<td>weight for sum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.15 branching/random

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/random/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>branching/random/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;random&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>branching/random/priority</td>
<td>priority of branching rule &lt;random&gt;</td>
<td>-100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>branching/random/seed</td>
<td>initial random seed value</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.16 branching/relpscost

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/relpscost/initcand</td>
<td>maximal number of candidates initialized with strong branching per node</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/inititer</td>
<td>iteration limit for strong branching initializations of pseudo cost entries (0: auto)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;relpscost&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/priority</td>
<td>priority of branching rule &lt;relpscost&gt;</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/sbiterofs</td>
<td>additional number of allowed strong branching LP iterations</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>branching/relpscost/sbiterquot</td>
<td>maximal fraction of strong branching LP iterations compared to node relaxation LP iterations</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/relpscost/confidencelevel</td>
<td>the confidence level for statistical methods, between 0 (Min) and 4 (Max). Range: {0, ..., 4}</td>
<td>2</td>
</tr>
<tr>
<td>branching/relpscost/conflictlengthweight</td>
<td>weight in score calculations for conflict length score</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/conflictweight</td>
<td>weight in score calculations for conflict score</td>
<td>0.01</td>
</tr>
<tr>
<td>branching/relpscost/cutoffweight</td>
<td>weight in score calculations for cutoff score</td>
<td>0.0001</td>
</tr>
<tr>
<td>branching/relpscost/degeneracyaware</td>
<td>should degeneracy be taken into account to update weights and skip strong branching? (0: off, 1: after root, 2: always) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/dynamicweights</td>
<td>should the weights of the branching rule be adjusted dynamically during solving based on objective and infeasible leaf counters? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/filtercandssym</td>
<td>Use symmetry to filter branching candidates?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/higherrortol</td>
<td>high relative error tolerance for reliability Range: ([0, \infty])</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/inferenceweight</td>
<td>weight in score calculations for inference score</td>
<td>0.0001</td>
</tr>
<tr>
<td>branching/relpscost/lowerrortol</td>
<td>low relative error tolerance for reliability Range: ([0, \infty])</td>
<td>0.05</td>
</tr>
<tr>
<td>branching/relpscost/maxbdchgs</td>
<td>maximal number of bound tightenings before the node is reevaluated (-1: unlimited) Range: (-1, ..., \infty)</td>
<td>5</td>
</tr>
<tr>
<td>branching/relpscost/maxlookahead</td>
<td>maximal number of further variables evaluated without better score Range: ({1, ..., \infty})</td>
<td>9</td>
</tr>
<tr>
<td>branching/relpscost/maxproprounds</td>
<td>maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings) Range: (-2, ..., \infty)</td>
<td>-2</td>
</tr>
<tr>
<td>branching/relpscost/maxreliable</td>
<td>maximal value for minimum pseudo cost size to regard pseudo cost value as reliable Range: ([0, \infty])</td>
<td>5</td>
</tr>
<tr>
<td>branching/relpscost/minreliable</td>
<td>minimal value for minimum pseudo cost size to regard pseudo cost value as reliable Range: ([0, \infty])</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/nlscoreweight</td>
<td>weight in score calculations for nlcound score Range: real</td>
<td>0.1</td>
</tr>
<tr>
<td>branching/relpscost/probingbounds</td>
<td>should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>branching/relpscost/pscostweight</td>
<td>weight in score calculations for pseudo cost score</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/randinitorder</td>
<td>should candidates be initialized in randomized order?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/skipbadinitcands</td>
<td>should branching rule skip candidates that have a low probability to be better than the best strong-branching or pseudo-candidate?</td>
<td>1</td>
</tr>
<tr>
<td>branching/relpscost/startrandseed</td>
<td>start seed for random number generation</td>
<td>5</td>
</tr>
<tr>
<td>branching/relpscost/storesemiinitcosts</td>
<td>should strong branching result be considered for pseudo costs if the other direction was infeasible?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/transsympscost</td>
<td>Transfer pscost information to symmetric variables?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/usedynamicconfidence</td>
<td>should the confidence level be adjusted dynamically?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/usehyptestforreliability</td>
<td>should the strong branching decision be based on a hypothesis test?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/usedereerrorreliability</td>
<td>should reliability be based on relative errors?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/usesblocalinfo</td>
<td>should the scoring function use only local cutoff and inference information obtained for strong branching candidates?</td>
<td>0</td>
</tr>
<tr>
<td>branching/relpscost/usesmallweightsitlim</td>
<td>should smaller weights be used for pseudo cost updates after hitting the LP iteration limit?</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.17 branching/treemodel

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/treemodel/enable</td>
<td>should candidate branching variables be scored using the Treemodel branching rules?</td>
<td>0</td>
</tr>
<tr>
<td>branching/treemodel/height</td>
<td>estimated tree height at which we switch from using the low rule to the high rule</td>
<td>10</td>
</tr>
<tr>
<td>branching/treemodel/highrule</td>
<td>scoring function to use at nodes predicted to be high in the tree ('d'efault, 's'vts, 'r'atio, 't'ree sample)</td>
<td>r</td>
</tr>
<tr>
<td>branching/treemodel/lowrule</td>
<td>scoring function to use at nodes predicted to be low in the tree ('d'efault, 's'vts, 'r'atio, 't'ree sample)</td>
<td>r</td>
</tr>
</tbody>
</table>
### 5.36 SCIP

#### 5.36.4.18 branching/vanillafullstrong

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| branching/vanillafullstrong/idempotent | should strong branching side-effects be prevented (e.g., domain changes, stat updates etc.)?  
Range: boolean                                                                                                         | 0       |
| branching/vanillafullstrong/integralcands | should integral variables in the current LP solution be considered as branching candidates?  
Range: boolean                                                                                                         | 0       |
| branching/vanillafullstrong/maxbounddist | maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)  
Range: [0, 1]                                                                                                         | 1       |
| branching/vanillafullstrong/maxdepth | maximal depth level, up to which branching rule <vanillafullstrong> should be used (-1 for no limit)  
Range: {-1, ..., 65534}                                                                                           | -1      |
| branching/vanillafullstrong/priority | priority of branching rule <vanillafullstrong>  
Range: {-536870912, ..., 536870911}                                                                                     | -2000   |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| branching/vanillafullstrong/collectscores | should strong branching scores be collected?  
Range: boolean | 0       |
| branching/vanillafullstrong/donotbranch    | should candidates only be scored, but no branching be performed?  
Range: boolean | 0       |
| branching/vanillafullstrong/scoreall      | should strong branching scores be computed for all candidates, or can we early stop when a variable has infinite score?  
Range: boolean | 0       |

5.36.4.19 conflict

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| conflict/conflictgraphweight       | the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict graph  
Range: [0, 1] | 1       |
| conflict/conflictweight            | the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict graph  
Range: [0, 1] | 0       |
| conflict/enable                    | should conflict analysis be enabled?  
Range: boolean | 1       |
| conflict/preferbinary              | should binary conflicts be preferred?  
Range: boolean | 0       |
| conflict/restartfac                | factor to increase restartnum with after each restart  
Range: [0, ∞] | 1.5     |
| conflict/restartnum                | number of successful conflict analysis calls that trigger a restart (0: disable conflict restarts)  
Range: {0, ..., ∞} | 0       |
| conflict/sepaaltproofs             | apply cut generating functions to construct alternative proofs  
Range: boolean | 0       |
| conflict/useboundlp                | should bound exceeding LP conflict analysis be used?  
('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray)  
Range: {o, c, d, b} | b       |
| conflict/useinflp                   | should infeasible LP conflict analysis be used?  
('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray)  
Range: {o, c, d, b} | b       |
| conflict/useprop                   | should propagation conflict analysis be used?  
Range: boolean | 1       |
| conflict/usepseudo                 | should pseudo solution conflict analysis be used?  
Range: boolean | 1       |
| conflict/usesb                     | should infeasible/bound exceeding strong branching conflict analysis be used?  
Range: boolean | 1       |

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| conflict/allowlocal                | should conflict constraints be generated that are only valid locally?  
Range: boolean | 1       |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/cleanboundexceedings</td>
<td>should conflicts based on an old cutoff bound be removed from the conflict pool after improving the primal bound?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/downlockscorefac</td>
<td>score factor for down locks in bound relaxation heuristic</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>conflict/dynamic</td>
<td>should the conflict constraints be subject to aging?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/fuiplevels</td>
<td>number of depth levels up to which first UIP’s are used in conflict analysis (-1: use All-FirstUIP rule)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/fullshortenconflict</td>
<td>try to shorten the whole conflict set or terminate early (depending on the 'maxvarsdetectimpliedbounds' parameter)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conflict/ignorerelaxedbd</td>
<td>should relaxed bounds be ignored?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/interconss</td>
<td>maximal number of intermediate conflict constraints generated in conflict graph (-1: use every intermediate constraint)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/keepreprop</td>
<td>should constraints be kept for repropagation even if they are too long?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/lpiterations</td>
<td>maximal number of LP iterations in each LP resolving loop (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/maxconss</td>
<td>maximal number of conflict constraints accepted at an infeasible node (-1: use all generated conflict constraints)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/maxlploops</td>
<td>maximal number of LP resolving loops during conflict analysis (-1: no limit)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/maxstoresize</td>
<td>maximal size of conflict store (-1: auto, 0: disable storage)</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/maxvarsdetectimpliedbounds</td>
<td>maximal number of variables to try to detect global bound implications and shorten the whole conflict set (0: disabled)</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>conflict/maxvarsfac</td>
<td>maximal fraction of variables involved in a conflict constraint</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>conflict/minimprove</td>
<td>minimal improvement of primal bound to remove conflicts based on a previous incumbent</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>conflict/minmaxvars</td>
<td>minimal absolute maximum of variables involved in a conflict constraint</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>conflict/prefinfproof</td>
<td>prefer infeasibility proof to boundexceeding proof</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/proofscorefac</td>
<td>score factor for impact on acticity in bound relaxation heuristic</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>conflict/reconvlevels</td>
<td>number of depth levels up to which UIP reconvergence constraints are generated (-1: generate reconvergence constraints in all depth levels)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>conflict/removable</td>
<td>should the conflict’s relaxations be subject to LP aging and cleanup?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/repropagate</td>
<td>should earlier nodes be repropogated in order to replace branching decisions by deductions?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/scorefac</td>
<td>factor to decrease importance of variables' earlier conflict scores</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-06, 1]</td>
<td></td>
</tr>
<tr>
<td>conflict/separate</td>
<td>should the conflict constraints be separated?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/settlelocal</td>
<td>should conflict constraints be attached only to the local subtree where they can be useful?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/uplocksorefac</td>
<td>score factor for up locks in bound relaxation heuristic</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>conflict/uselocalrows</td>
<td>use local rows to construct infeasibility proofs</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>conflict/weightrepropdepth</td>
<td>weight of the repropagation depth of a conflict used in score calculation</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>conflict/weightsize</td>
<td>weight of the size of a conflict used in score calculation</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>conflict/weightvaliddepth</td>
<td>weight of the valid depth of a conflict used in score calculation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.20 **conflict/bounddisjunction**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/bounddisjunction/continuousfrac</td>
<td>maximal percentage of continuous variables within a conflict</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conflict/bounddisjunction/priority</td>
<td>priority of conflict handler (&lt;\text{bounddisjunction}&gt;)</td>
<td>-3000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-2147483648, ..., \infty}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.21 **conflict/graph**
5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>conflict/graph/depthscorefac</td>
<td>score factor for depth level in bound relaxation heuristic</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.22 conflict/indicatorconflict

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>conflict/indicatorconflict/priority</td>
<td>priority of conflict handler &lt;indicatorconflict&gt;</td>
<td>200000</td>
</tr>
<tr>
<td></td>
<td>Range: {-2147483648, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.23 conflict/linear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>conflict/linear/priority</td>
<td>priority of conflict handler &lt;linear&gt;</td>
<td>-1000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-2147483648, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.24 conflict/logicor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>conflict/logicor/priority</td>
<td>priority of conflict handler &lt;logicor&gt;</td>
<td>800000</td>
</tr>
<tr>
<td></td>
<td>Range: {-2147483648, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.25 conflict/setppc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>conflict/setppc/priority</td>
<td>priority of conflict handler &lt;setppc&gt;</td>
<td>700000</td>
</tr>
<tr>
<td></td>
<td>Range: {-2147483648, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.26 constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>constraints/age_limit</td>
<td>maximum age an unnecessary constraint can reach before it is deleted (0: dynamic, -1: keep all constraints)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/disableenfops</td>
<td>should enforcement of pseudo solution be disabled?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/obsoleteage</td>
<td>age of a constraint after which it is marked obsolete (0: dynamic, -1 do not mark constraints obsolete)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.27 constraints/SOS1

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS1/branchnonzeros</td>
<td>Branch on SOS constraint with most number of nonzeros?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/branchsos</td>
<td>Use SOS1 branching in enforcing (otherwise leave decision to branching rules)? This value can only be set to false if all SOS1 variables are binary</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/branchweight</td>
<td>Branch on SOS cons. with highest nonzero-variable weight for branching (needs branchnonzeros = false)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/seapfreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS1/addbdsfeas</td>
<td>minimal feasibility value for bound inequalities in order to be added to the branching node</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/addcomps</td>
<td>if TRUE then add complementarity constraints to the branching nodes (can be used in combination with neighborhood or bipartite branching)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/addcompsdepth</td>
<td>only add complementarity constraints to branching nodes for predefined depth (-1: no limit)</td>
<td>30</td>
</tr>
<tr>
<td>constraints/SOS1/addcompsfeas</td>
<td>minimal feasibility value for complementarity constraints in order to be added to the branching node</td>
<td>-0.6</td>
</tr>
<tr>
<td>constraints/SOS1/addextendedbds</td>
<td>should added complementarity constraints be extended to SOS1 constraints to get tighter bound inequalities</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/autocutsfromsos1</td>
<td>if TRUE then automatically switch to separating initial SOS1 constraints if the SOS1 constraints do not overlap</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/autosos1branch</td>
<td>if TRUE then automatically switch to SOS1 branching if the SOS1 constraints do not overlap</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/boundcutsdepth</td>
<td>node depth of separating bound cuts (-1: no limit)</td>
<td>40</td>
</tr>
<tr>
<td>constraints/SOS1/boundcutsfreq</td>
<td>frequency for separating bound cuts; zero means to separate only in the root node</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/SOS1/boundcutsfromgraph</td>
<td>if TRUE separate bound inequalities from the conflict graph</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/boundcutsfromsos1</td>
<td>if TRUE separate bound inequalities from initial SOS1 constraints</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/branchingrule</td>
<td>which branching rule should be applied? ('n': neighborhood, 'b': bipartite, 's': SOS1/clique) (note: in some cases an automatic switching to SOS1 branching is possible)</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>Range: n, b, s</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/conflictprop</td>
<td>whether to use conflict graph propagation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/depthimplanalyzer</td>
<td>number of recursive calls of implication graph analysis (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, 65534}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/fixnonzero</td>
<td>if neighborhood branching is used, then fix the branching variable (if positive in sign) to the value of the feasibility tolerance</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/implcutsdepth</td>
<td>node depth of separating implied bound cuts (-1: no limit)</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/implcutsfreq</td>
<td>frequency for separating implied bound cuts; zero means to separate only in the root node</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, 65534}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/implprop</td>
<td>whether to use implication graph propagation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxaddcomps</td>
<td>maximal number of complementarity constraints added per branching node (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxboundcuts</td>
<td>maximal number of bound cuts separated per branching node</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: $${0, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxboundcutsroot</td>
<td>maximal number of bound cuts separated per iteration in the root node</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Range: $${0, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxextensions</td>
<td>maximal number of extensions that will be computed for each SOS1 constraint (-1: no limit)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: $${-1, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/implcuts</td>
<td>maximal number of implied bound cuts separated per branching node</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: $${0, \ldots, \infty}$$</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/SOS1/maximplcutsroot</td>
<td>maximal number of implied bound cuts separated per iteration in the root node</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Range: ( {0, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in ((-1: \text{no limit}))</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-1, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxsosadjacency</td>
<td>do not create an adjacency matrix if number of SOS1 variables is larger than predefined value ((-1: \text{no limit}))</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-1, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/maxtightenbds</td>
<td>maximal number of bound tightening rounds per presolving round ((-1: \text{no limit}))</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-1, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/nstrongiter</td>
<td>maximal number LP iterations to perform for each strong branching round ((-2: \text{auto}, -1: \text{no limit}))</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-2, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/nstrongrounds</td>
<td>maximal number of strong branching rounds to perform for each node ((-1: \text{auto}); only available for neighborhood and bipartite branching)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ( {-1, ..., \infty} )</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/perfimplanalysis</td>
<td>if TRUE then perform implication graph analysis (might add additional SOS1 constraints)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/presoltiming</td>
<td>timing mask of the constraint handler’s presolving method ((4: \text{FAST}, 8: \text{MEDIUM}, 16: \text{EXHAUSTIVE}, 32: \text{FINAL}))</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/proptiming</td>
<td>timing when constraint propagation should be called ((1: \text{BEFORELP}, 2: \text{DURINGLPLOOP}, 4: \text{AFTERLPLOOP}, 15: \text{ALWAYS}))</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/sosconsprop</td>
<td>whether to use SOS1 constraint propagation</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS1/strthenboundcuts</td>
<td>if TRUE then bound cuts are strengthened in case bound variables are available</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.28 constraints/SOS2

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS2/propfreq</td>
<td>frequency for propagating domains ((-1: \text{never}, 0: \text{only in root node}))</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/SOS2/sepafreq</td>
<td>frequency for separating cuts ((-1: \text{never}, 0: \text{only in root node}))</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS2/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/SOS2/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS2/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, ..., 65534]</td>
<td>100</td>
</tr>
<tr>
<td>constraints/SOS2/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ..., ∞]</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/SOS2/presoltiming</td>
<td>timing mask of the constraint handler’s presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>4</td>
</tr>
<tr>
<td>constraints/SOS2/proptiming</td>
<td>timing when constraint propagation should be called (1:BEBORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.29 constraints/and

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/and/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, ..., 65534]</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, ..., 65534]</td>
<td>1</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/and/aggrlinearization</td>
<td>should an aggregated linearization be used? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/dualpresolving</td>
<td>should dual presolving be performed? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, ..., 65534]</td>
<td>100</td>
</tr>
<tr>
<td>constraints/and/enforcecuts</td>
<td>should cuts be separated during LP enforcing? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/linearize</td>
<td>should the AND-constraint get linearized and removed (in presolving)? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ..., ∞]</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/and/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>20</td>
</tr>
<tr>
<td>constraints/and/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/upgraderesultant</td>
<td>should all binary resultant variables be upgraded to implicit binary variables? Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.30 constraints/bounddisjunction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/bounddisjunction/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bounddisjunction/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/bounddisjunction/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bounddisjunction/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bounddisjunction/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}</td>
<td>100</td>
</tr>
<tr>
<td>constraints/bounddisjunction/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/bounddisjunction/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>4</td>
</tr>
<tr>
<td>constraints/bounddisjunction/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.36.4.31 constraints/components

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/components/intfactor</td>
<td>the weight of an integer variable compared to binary variables</td>
<td>1</td>
</tr>
<tr>
<td>constraints/components/maxdepth</td>
<td>maximum depth of a node to run components detection (-1: disable component detection during solving)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/components/nodelimit</td>
<td>maximum number of nodes to be solved in subproblems during presolving</td>
<td>10000</td>
</tr>
<tr>
<td>constraints/components/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/components/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/components/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/components/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/components/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/components/feastolfactor</td>
<td>factor to increase the feasibility tolerance of the main SCIP in all sub-SCIPs, default value 1.0</td>
<td>1</td>
</tr>
<tr>
<td>constraints/components/maxintvars</td>
<td>maximum number of integer (or binary) variables to solve a subproblem during presolving (-1: unlimited)</td>
<td>500</td>
</tr>
<tr>
<td>constraints/components/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/components/minrelsize</td>
<td>minimum relative size (in terms of variables) to solve a component individually during branch-and-bound</td>
<td>0.1</td>
</tr>
<tr>
<td>constraints/components/minsize</td>
<td>minimum absolute size (in terms of variables) to solve a component individually during branch-and-bound</td>
<td>50</td>
</tr>
<tr>
<td>constraints/components/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>32</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/components/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/maxsepacuts</td>
<td>maximal number of cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td>constraints/indicator/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node</td>
<td>2000</td>
</tr>
<tr>
<td>constraints/indicator/maxsepanonviolated</td>
<td>maximal number of separated non violated IISs, before separation is stopped</td>
<td>3</td>
</tr>
<tr>
<td>constraints/indicator/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/indicator/addcoupling</td>
<td>Add coupling constraints or rows if big-M is small enough?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/addcouplingcons</td>
<td>Add initial variable upper bound constraints, if 'addcoupling' is true?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/addopposite</td>
<td>Add opposite inequality in nodes in which the binary variable has been fixed to 0?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/branchindicators</td>
<td>Branch on indicator constraints in enforcing?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/conflictsupgrade</td>
<td>Try to upgrade bounddisjunction conflicts by replacing slack variables?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/dualreductions</td>
<td>Should dual reduction steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/indicator/enforcecuts</td>
<td>In enforcing try to generate cuts (only if sepaalternativelp is true)? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/forcerestart</td>
<td>Force restart if absolute gap is 1 or enough binary variables have been fixed? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/generatebilinear</td>
<td>Do not generate indicator constraint, but a bilinear constraint instead? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/genlogicor</td>
<td>Generate logicor constraints instead of cuts? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/maxconditionaltlp</td>
<td>maximum estimated condition of the solution basis matrix of the alternative LP to be trustworthy (0.0 to disable check) Range: [0, \infty]</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/maxcouplingvalue</td>
<td>maximum coefficient for binary variable in coupling constraint Range: [0, 1e+09]</td>
<td>10000</td>
</tr>
<tr>
<td>constraints/indicator/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/indicator/nolinconscont</td>
<td>Decompose problem (do not generate linear constraint if all variables are continuous)? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>4</td>
</tr>
<tr>
<td>constraints/indicator/proptiming</td>
<td>timing when constraint propagation should be called {1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS} Range: {1, ..., 15}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/removeindicators</td>
<td>Remove indicator constraint if corresponding variable bound constraint has been added? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/restartfrac</td>
<td>fraction of binary variables that need to be fixed before restart occurs (in forcerestart) Range: [0, 1]</td>
<td>0.9</td>
</tr>
<tr>
<td>constraints/indicator/scaleslackvar</td>
<td>Scale slack variable coefficient at construction time? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepaalternativelp</td>
<td>Separate using the alternative LP? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplingcuts</td>
<td>Should the coupling inequalities be separated dynamically? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplinglocal</td>
<td>Allow to use local bounds in order to separate coupling inequalities? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplingvalue</td>
<td>maximum coefficient for binary variable in separated coupling constraint</td>
<td>10000</td>
</tr>
<tr>
<td>constraints/indicator/sepaperspective</td>
<td>Separate cuts based on perspective formulation?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepapersplocal</td>
<td>Allow to use local bounds in order to separate perspective cuts?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/trysolfromcover</td>
<td>Try to construct a feasible solution from a cover?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/trysolutions</td>
<td>Try to make solutions feasible by setting indicator variables?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/updatebounds</td>
<td>Update bounds of original variables for separation?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/upgradelinear</td>
<td>Try to upgrade linear constraints to indicator constraints?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/useobjectivecut</td>
<td>Use objective cut with current best solution to alternative LP?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/useotherconss</td>
<td>Collect other constraints to alternative LP?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.33 constraints/integral

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/integral/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/integral/sepa</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/integral/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/integral/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/integral/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td>constraints/integral/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.34 constraints/knapsack

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/knapsack/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}</td>
<td>5</td>
</tr>
<tr>
<td>constraints/knapsack/maxroundsroot</td>
<td>maximal number of separation rounds per node in the root node (-1: unlimited) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/knapsack/maxsepacuts</td>
<td>maximal number of cuts separated per separation round Range: {0, ..., ∞}</td>
<td>50</td>
</tr>
<tr>
<td>constraints/knapsack/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node Range: {0, ..., ∞}</td>
<td>200</td>
</tr>
<tr>
<td>constraints/knapsack/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>0</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/knapsack/cliqueextractfactor</td>
<td>lower clique size limit for greedy clique extraction algorithm (relative to largest clique) Range: [0, 1]</td>
<td>0.5</td>
</tr>
<tr>
<td>constraints/knapsack/clqpartupdatefac</td>
<td>factor on the growth of global cliques to decide when to update a previous (negated) clique partition (used only if updatecliquepartitions is set to TRUE) Range: [1, 10]</td>
<td>1.5</td>
</tr>
<tr>
<td>constraints/knapsack/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/detectcutoffbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/knapsack/detectlowerbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/disaggregation</td>
<td>should disaggregation of knapsack constraints be allowed in preprocessing?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/dualpresolving</td>
<td>should dual presolving steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/knapsack/maxcardbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cuts</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/knapsack/negatedclique</td>
<td>should negated clique information be used in solving process</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td>constraints/knapsack/presolvehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/sepacardfreq</td>
<td>multiplier on separation frequency, how often knapsack cuts are separated (-1: never, 0: only at root)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/simplifyinequalities</td>
<td>should presolving try to simplify knapsacks</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/updatecliquepartitions</td>
<td>should clique partition information be updated when old partition seems outdated?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/usegubs</td>
<td>should GUB information be used for separation?</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.36.4.35 constraints/linear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/linear/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>constraints/linear/maxroundsroot</td>
<td>maximal number of separation rounds per node in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/linear/maxsepacuts</td>
<td>maximal number of cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>constraints/linear/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node</td>
<td>200</td>
</tr>
<tr>
<td>constraints/linear/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/separateall</td>
<td>should all constraints be subject to cardinality cut generation instead of only the ones with non-zero dual value?</td>
<td>0</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/linear/aggregatevariables</td>
<td>should presolving search for aggregations in equations</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/checkrelmaxabs</td>
<td>should the violation for a constraint with side 0.0 be checked relative to 1.0 (FALSE) or to the maximum absolute value in the activity (TRUE)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/detectcutoffbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/detectlowerbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/linear/detectpartialobjective</td>
<td>should presolving try to detect subsets of constraints parallel to the objective function? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/dualpresolving</td>
<td>should dual presolving steps be performed? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}</td>
<td>100</td>
</tr>
<tr>
<td>constraints/linear/extractcliques</td>
<td>should Cliques be extracted? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/maxaggnormsclae</td>
<td>maximal allowed relative gain in maximum norm for constraint aggregation (0.0: disable constraint aggregation) Range: ([0, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/maxcardbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cardinality cuts Range: ([0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/maxdualmultaggrquot</td>
<td>maximum coefficient dynamism (ie. maxabsval / minabsval) for dual multiaggregation Range: ([1, \infty])</td>
<td>(\infty)</td>
</tr>
<tr>
<td>constraints/linear/maxeasyactivitydelta</td>
<td>maximum activity delta to run easy propagation on linear constraint (faster, but numerically less stable) Range: ([0, \infty])</td>
<td>1e+06</td>
</tr>
<tr>
<td>constraints/linear/maxmultaggrquot</td>
<td>maximum coefficient dynamism (ie. maxabsval / minabsval) for primal multiaggregation Range: ([1, \infty])</td>
<td>1000</td>
</tr>
<tr>
<td>constraints/linear/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: ([-1, ..., \infty])</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/linear/mingainpermincomparisons</td>
<td>minimal gain per minimal pairwise presolve comparisons to repeat pairwise comparison round Range: ([0, 1])</td>
<td>1e-06</td>
</tr>
<tr>
<td>constraints/linear/multaggrremove</td>
<td>should multi-aggregations only be performed if the constraint can be removed afterwards? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/nmincomparisons</td>
<td>number for minimal pairwise presolve comparisons Range: ([1, ..., \infty])</td>
<td>200000</td>
</tr>
<tr>
<td>constraints/linear/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/linear/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8: MEDIUM, 16: EXHAUSTIVE, 32: FINAL) Range: {4, ..., 60}</td>
<td>20</td>
</tr>
<tr>
<td>constraints/linear/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/proptiming</td>
<td>timing when constraint propagation should be called (1: BEFORELP, 2: DURING LPLOOP, 4: AFTERLPLOOP, 15: ALWAYS) Range: {1, ..., 15}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/rangedrowartcons</td>
<td>should presolving and propagation extract sub-constraints from ranged rows and equations? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/rangedrowfreq</td>
<td>frequency for applying ranged row propagation Range: {1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/rangedrowmaxdepth</td>
<td>maximum depth to apply ranged row propagation Range: {0, ..., ∞}</td>
<td>∞</td>
</tr>
<tr>
<td>constraints/linear/rangedrowpropagation</td>
<td>should presolving and propagation try to improve bounds, detect infeasibility, and extract sub-constraints from ranged rows and equations? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/simplifyinequalities</td>
<td>should presolving try to simplify inequalities Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/singletonstuffing</td>
<td>should stuffing of singleton continuous variables be performed? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/singlevarstuffing</td>
<td>should single variable stuffing be performed, which tries to fulfill constraints using the cheapest variable? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/sortvars</td>
<td>apply binaries sorting in decr. order of coeff abs value? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/tightenboundsfreq</td>
<td>multiplier on propagation frequency, how often the bounds are tightened (-1: never, 0: only at root) Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.36 constraints/linear/upgrade

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/linear/upgrade/indicator</td>
<td>enable linear upgrading for constraint handler &lt;indicator&gt; Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/knapsack</td>
<td>enable linear upgrading for constraint handler &lt;knapsack&gt; Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/linear/upgrade/logicor</td>
<td>enable linear upgrading for constraint handler <code>&lt;logicor&gt;</code>&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/setppc</td>
<td>enable linear upgrading for constraint handler <code>&lt;setppc&gt;</code>&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/varbound</td>
<td>enable linear upgrading for constraint handler <code>&lt;varbound&gt;</code>&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/xor</td>
<td>enable linear upgrading for constraint handler <code>&lt;xor&gt;</code>&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.37 constraints/logicor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/logicor/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)&lt;br&gt;Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/separfreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)&lt;br&gt;Range: {-1, ..., 65534}</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/logicor/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/dualpresolving</td>
<td>should dual presolving steps be performed?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)&lt;br&gt;Range: {-1, ..., 65534}</td>
<td>100</td>
</tr>
<tr>
<td>constraints/logicor/implications</td>
<td>should implications/cliques be used in presolving&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)&lt;br&gt;Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/logicor/negatedclique</td>
<td>should negated clique information be used in presolving&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/presoltiming</td>
<td>timing mask of the constraint handler’s presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)&lt;br&gt;Range: {4, ..., 60}</td>
<td>28</td>
</tr>
</tbody>
</table>
### Option Descriptions

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/logicor/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/logicor/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
<tr>
<td>constraints/logicor/strengthen</td>
<td>should pairwise constraint comparison try to strengthen constraints by removing superfluous non-zeros?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/assumeconvex</td>
<td>whether to assume that any constraint is convex</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/bilinmaxnauxexpressions</td>
<td>maximal number of auxiliary expressions per bilinear term</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/linearizeheursol</td>
<td>whether tight linearizations of nonlinear constraints should be added to cutpool when some heuristics finds a new solution ('o'off, on new 'i'ncumbents, on 'e'very solution)</td>
<td>i if QCP or NLP, o otherwise</td>
</tr>
<tr>
<td></td>
<td>Range: o, i, e</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/maxproprounds</td>
<td>limit on number of propagation rounds for a set of constraints within one round of SCIP propagation</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/reformbinprods</td>
<td>whether to reformulate products of binary variables during presolving</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/reformbinprodsand</td>
<td>whether to use the AND constraint handler for reformulating binary products</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/reformbinprodsfac</td>
<td>minimum number of terms to reformulate bilinear binary products by factorizing variables (\leq 1: \text{disabled})</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Options for expert users</td>
<td>whether variables contained in a single constraint should be forced to be at their lower or upper bounds ('d'isable, change 't'ype, add 'b'ound disjunction)</td>
<td>t</td>
</tr>
<tr>
<td></td>
<td>by how much to relax constraint sides during bound tightening</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>an expression will be enforced if the &quot;auxiliary&quot; violation is at least this factor times the &quot;original&quot; violation</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>whether to forbid multiaggregation of nonlinear variables</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>whether to force &quot;strong&quot; cuts in enforcement</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>whether to check bounds of all auxiliary variable to seed reverse propagation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>whether to (re)run propagation in enforcement</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/nonlinear/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/rownotremovable</td>
<td>whether to make rows to be non-removable in the node where they are added (can prevent some cycling): 'off', in enforcement only, 'always'</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/strongcutefficacy</td>
<td>consider efficacy requirement when deciding whether a cut is &quot;strong&quot;</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/strongcutmaxcoef</td>
<td>&quot;strong&quot; cuts will be scaled to have their maximal coef in [1/strongcutmaxcoef, strongcutmaxcoef]</td>
<td>1000</td>
</tr>
<tr>
<td>constraints/nonlinear/tightenlpfeastol</td>
<td>whether to tighten LP feasibility tolerance during enforcement, if it seems useful</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/varboundrelax</td>
<td>strategy on how to relax variable bounds during bound tightening: relax (n)ot, relax by (a)bolute value, relax always by (a)bolute value, relax by (r)relative value</td>
<td>r</td>
</tr>
<tr>
<td>constraints/nonlinear/varboundrelaxamount</td>
<td>by how much to relax variable bounds during bound tightening if strategy 'a', 'b', or 'r'</td>
<td>1e-09</td>
</tr>
<tr>
<td>constraints/nonlinear/violscale</td>
<td>method how to scale violations to make them comparable (not used for feasibility check): (n)one, (a)ctivity and side, norm of (g)radient</td>
<td>n</td>
</tr>
<tr>
<td>constraints/nonlinear/vpadjfacetthresh</td>
<td>adjust computed facet of envelope of vertex-polyhedral function up to a violation of this value times LP feasibility tolerance</td>
<td>10</td>
</tr>
<tr>
<td>constraints/nonlinear/vpdualsimplex</td>
<td>whether to use dual simplex instead of primal simplex for LP that computes facet of vertex-polyhedral function</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/nonlinear/vpmaxperturb</td>
<td>maximal relative perturbation of reference point when computing facet of envelope of vertex-polyhedral function (dim $\geq 2$)</td>
<td>0.001</td>
</tr>
<tr>
<td>constraints/nonlinear/weakcutminviolfactor</td>
<td>retry enforcing of constraint with weak cuts if violation is least this factor of maximal violated constraints</td>
<td>0.5</td>
</tr>
<tr>
<td>constraints/nonlinear/weakcutthreshold</td>
<td>threshold for when to regard a cut from an estimator as weak (lower values allow more weak cuts)</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### 5.36.4.39 constraints/nonlinear/branching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/nonlinear/branching/aux</td>
<td>from which depth on in the tree to allow branching on auxiliary variables (variables added for extended formulation)</td>
<td>$\infty$</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/domainweight</td>
<td>weight by how much to consider the domain width in branching score</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/dualweight</td>
<td>weight by how much to consider the dual values of rows that contain a variable for its branching score</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/external</td>
<td>whether to use external branching candidates and branching rules for branching</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/highscorefactor</td>
<td>consider a variable branching score high if its branching score $\geq$ this factor $\times$ maximal branching score among all variables</td>
<td>0.9</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/highviolfactor</td>
<td>consider a constraint highly violated if its violation is $\geq$ this factor $\times$ maximal violation among all constraints</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/pscostreliable</td>
<td>minimum pseudo-cost update count required to consider pseudo-costs reliable</td>
<td>2</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/pscostweight</td>
<td>weight by how much to consider the pseudo cost of a variable for its branching score</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/scoreagg</td>
<td>how to aggregate several branching scores given for the same expression: 'a'verage, 'm'aximum, 's'um</td>
<td>s</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/vartypeweight</td>
<td>weight by how much to consider variable type (continuous: 0, binary: 1, integer: 0.1, impl-integer: 0.01) in branching score</td>
<td>0.5</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/violsplit</td>
<td>method used to split violation in expression onto variables: 'u'iform, 'm'idness of solution, 'd'omain width, 'l'ogarithmic domain width</td>
<td>m</td>
</tr>
<tr>
<td>constraints/nonlinear/branching/violweight</td>
<td>weight by how much to consider the violation assigned to a variable for its branching score</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.40 constraints/nonlinear/upgrade

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/nonlinear/upgrade/linear</td>
<td>enable nonlinear upgrading for constraint handler &lt;linear&gt;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/nonlinear/upgrade/setppc</td>
<td>enable nonlinear upgrading for constraint handler &lt;setppc&gt;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.41 constraints/or

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/or/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/or/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constraints/or/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/or/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/or/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/or/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/or/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.42 constraints/orbisack

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/orbisack/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/sepfreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/checkpporbisack</td>
<td>Upgrade orbisack constraints to packing/partitioning orbisacks?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/coeffbound</td>
<td>Maximum size of coefficients for orbisack inequalities</td>
<td>1e+06</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/coverseparation</td>
<td>Separate cover inequalities for orbisacks?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/forceconscopy</td>
<td>Whether orbisack constraints should be forced to be copied to sub SCIPs.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/orbiSeparation</td>
<td>Separate orbisack inequalities?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbisack/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.43 constraints/orbitope

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/orbitope/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/orbitope/checkpporbiotope</td>
<td>Strengthen orbitope constraints to packing/partitioning orbitopes?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/forceconscopy</td>
<td>Whether orbitope constraints should be forced to be copied to sub SCIPs.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
<tr>
<td>constraints/orbitope/sepafullorbitope</td>
<td>Whether we separate inequalities for full orbitopes?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.44 constraints/setppc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/setppc/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/setppc/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/setppc/addvariablesascliques</td>
<td>should we try to generate extra cliques out of all binary variables to maybe fasten redundant constraint detection&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/cliquelifting</td>
<td>should we try to lift variables into other clique constraints, fix variables, aggregate them, and also shrink the amount of variables in clique constraints&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/cliqueshrinking</td>
<td>should we try to shrink the number of variables in a clique constraints, by replacing more than one variable by only one&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/setppc/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/dualpresolving</td>
<td>should dual presolving steps be performed?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/setppc/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)&lt;br&gt;Range: {-1, ..., 65534}</td>
<td>100</td>
</tr>
<tr>
<td>constraints/setppc/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)&lt;br&gt;Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/setppc/npseudobranches</td>
<td>number of children created in pseudo branching (0: disable pseudo branching)&lt;br&gt;Range: {0, ..., ∞}</td>
<td>2</td>
</tr>
<tr>
<td>constraints/setppc/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/setppc/presol.timing</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)&lt;br&gt;Range: {4, ..., 60}</td>
<td>28</td>
</tr>
<tr>
<td>constraints/setppc/presol.hashing</td>
<td>should hash table be used for detecting redundant constraints in advance&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>constraints/setppc/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)&lt;br&gt;Range: {1, ..., 15}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.45 constraints/symresack
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/symresack/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/checkmonotonicity</td>
<td>Check whether permutation is monotone when upgrading to packing/partitioning symresacks?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/forceconscopy</td>
<td>Whether symresack constraints should be forced to be copied to sub SCIPs.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/ppsymresack</td>
<td>Upgrade symresack constraints to packing/partitioning symresacks?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>constraints/symresack/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.46 constraints/varbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/varbound/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/varbound/usebdwidening</td>
<td>should bound widening be used in conflict analysis?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/maxlpcoef</td>
<td>maximum coefficient in varbound constraint to be added as a row into LP</td>
<td>1e+09</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/presoltiming</td>
<td>timing mask of the constraint handler's presolving method (4:FAST, 8: MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>constraints/varbound/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2: DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>

**5.36.4.47 constraints/xor**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/xor/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>constraints/xor/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constraints/xor/addextendedform</td>
<td>should the extended formulation be added in presolving?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/xor/addflowextended</td>
<td>should the extended flow formulation be added (non-symmetric formulation otherwise)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>constraints/xor/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/xor/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/xor/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/xor/gausspropfreq</td>
<td>frequency for applying the Gauss propagator</td>
<td>5</td>
</tr>
<tr>
<td>constraints/xor/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/xor/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/xor/presoltiming</td>
<td>timing mask of the constraint handler’s presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td>constraints/xor/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/xor/proptiming</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/xor/separateparity</td>
<td>should parity inequalities be separated?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.48 cutselection/hybrid

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cutselection/hybrid/dircutoffdistweight</td>
<td>weight of directed cutoff distance in cut score calculation</td>
<td>0</td>
</tr>
<tr>
<td>cutselection/hybrid/efficacyweight</td>
<td>weight of efficacy in cut score calculation</td>
<td>1</td>
</tr>
<tr>
<td>cutselection/hybrid/intsupportweight</td>
<td>weight of integral support in cut score calculation</td>
<td>0.1</td>
</tr>
<tr>
<td>cutselection/hybrid/minortho</td>
<td>minimal orthogonality for a cut to enter the LP</td>
<td>0.9</td>
</tr>
<tr>
<td>cutselection/hybrid/minorthoroot</td>
<td>minimal orthogonality for a cut to enter the LP in the root node</td>
<td>0.9</td>
</tr>
<tr>
<td>cutselection/hybrid/objparalweight</td>
<td>weight of objective parallelism in cut score calculation</td>
<td>0.1</td>
</tr>
<tr>
<td>cutselection/hybrid/priority</td>
<td>priority of cut selection rule &lt;hybrid&gt;</td>
<td>8000</td>
</tr>
</tbody>
</table>
### 5.36.4.49 decomposition

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>decomposition/disablemeasures</td>
<td>disable expensive measures&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>decomposition/maxgraphedge</td>
<td>maximum number of edges in block graph computation&lt;br&gt;(-1: no limit, 0: disable block graph computation)&lt;br&gt;Range: {-1, ..., ∞}</td>
<td>10000</td>
</tr>
</tbody>
</table>

### 5.36.4.50 display

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/allviols</td>
<td>display all violations for a given start solution / the best solution after the solving process?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>display/freq</td>
<td>frequency for displaying node information lines&lt;br&gt;Range: {-1, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>display/headerfreq</td>
<td>frequency for displaying header lines (every n'th node information line)&lt;br&gt;Range: {-1, ..., ∞}</td>
<td>15</td>
</tr>
<tr>
<td>display/lpinfo</td>
<td>should the LP solver display status messages?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>display/relevantstats</td>
<td>should the relevant statistics be displayed at the end of solving?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>display/statistics</td>
<td>whether to print statistics on a solve and a provided decomposition&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>display/verblevel</td>
<td>verbosity level of output&lt;br&gt;Range: {0, ..., 5}</td>
<td>4</td>
</tr>
<tr>
<td>display/width</td>
<td>maximal number of characters in a node information line&lt;br&gt;Range: {0, ..., ∞}</td>
<td>143 (80 for Windows without IDE)</td>
</tr>
</tbody>
</table>

### 5.36.4.51 display/avgdualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/avgdualbound/active</td>
<td>display activation status of display column &lt;avgdualbound&gt; (0: off, 1: auto, 2:on)&lt;br&gt;Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.52 display/completed
### 5.36.4.53 display/concdualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/concdualbound/active</code></td>
<td>display activation status of display column <code>&lt;concdualbound&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.54 display/concgap

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/concgap/active</code></td>
<td>display activation status of display column <code>&lt;concgap&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.55 display/concmemused

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/concmemused/active</code></td>
<td>display activation status of display column <code>&lt;concmemused&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.56 display/concprimalbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/concprimalbound/active</code></td>
<td>display activation status of display column <code>&lt;concprimalbound&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.57 display/concsolfound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/concsolfound/active</code></td>
<td>display activation status of display column <code>&lt;concsolfound&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.58 display/conflicts
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/conflicts/active</td>
<td>display activation status of display column &lt;conflicts&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.59  display/conss

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/conss/active</td>
<td>display activation status of display column &lt;conss&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.60  display/curcols

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/curcols/active</td>
<td>display activation status of display column &lt;curcols&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.61  display/curconss

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/curconss/active</td>
<td>display activation status of display column &lt;curconss&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.62  display/curdualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/curdualbound/active</td>
<td>display activation status of display column &lt;curdualbound&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.63  display/currows

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/currows/active</td>
<td>display activation status of display column &lt;currows&gt; (0: off, 1: auto, 2: on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.64  display/cutoffbound
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/cutoffbound/active</td>
<td>display activation status of display column <code>&lt;cutoffbound&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>display/cuts/active</td>
<td>display activation status of display column <code>&lt;cuts&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td>display/depth/active</td>
<td>display activation status of display column <code>&lt;depth&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td>display/dualbound/active</td>
<td>display activation status of display column <code>&lt;dualbound&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td>display/estimate/active</td>
<td>display activation status of display column <code>&lt;estimate&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td>display/feasST/active</td>
<td>display activation status of display column <code>&lt;feasST&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>0</td>
</tr>
<tr>
<td>display/gap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>display/gap/active</td>
<td>display activation status of display column (&lt;\text{gap}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.71 display/lpavgiterations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpavgiterations/active</td>
<td>display activation status of display column (&lt;\text{lpavgiterations}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1 (0 for Windows without IDE)</td>
</tr>
</tbody>
</table>

### 5.36.4.72 display/lpcond

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpcond/active</td>
<td>display activation status of display column (&lt;\text{lpcond}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.73 display/lpiterations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpiterations/active</td>
<td>display activation status of display column (&lt;\text{lpiterations}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.74 display/lpobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpobj/active</td>
<td>display activation status of display column (&lt;\text{lpobj}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.75 display/maxdepth

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/maxdepth/active</td>
<td>display activation status of display column (&lt;\text{maxdepth}&gt;) (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1 (0 for Windows without IDE)</td>
</tr>
</tbody>
</table>

### 5.36.4.76 display/memtotal
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/memtotal/active</td>
<td>display activation status of display column <code>&lt;memtotal&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.77 display/memused

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/memused/active</td>
<td>display activation status of display column <code>&lt;memused&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.78 display/nexternbranchcands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/nexternbranchcands/active</td>
<td>display activation status of display column <code>&lt;nexternbranchcands&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.79 display/nfrac

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/nfrac/active</td>
<td>display activation status of display column <code>&lt;nfrac&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.80 display/ninfeasleaves

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/ninfeasleaves/active</td>
<td>display activation status of display column <code>&lt;ninfeasleaves&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.81 display/nnodes

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/nnodes/active</td>
<td>display activation status of display column <code>&lt;nnodes&gt;</code> (0: off, 1: auto, 2:on) Range: {0, ..., 2}</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.82 display/nnodesbelowinc
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/nodesbelowinc/active` | display activation status of display column `<nodesbelowinc>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 0       |

5.36.4.83  `display/nobjleaves`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/nobjleaves/active` | display activation status of display column `<nobjleaves>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 1       |

5.36.4.84  `display/nodesleft`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/nodesleft/active` | display activation status of display column `<nodesleft>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 1       |

5.36.4.85  `display/nrank1nodes`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/nrank1nodes/active` | display activation status of display column `<nrank1nodes>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 0       |

5.36.4.86  `display/nsols`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/nsols/active` | display activation status of display column `<nsols>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 1       |

5.36.4.87  `display/plungedepth`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `display/plungedepth/active` | display activation status of display column `<plungedepth>` (0: off, 1: auto, 2:on)  
Range: {0, ..., 2} | 1       |

5.36.4.88  `display/poolsize`
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/poolsise/active</td>
<td>display activation status of display column &lt;poolsise&gt; (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.89 display/primalbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/primalbound/active</td>
<td>display activation status of display column &lt;primalbound&gt; (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.90 display/primalgap

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/primalgap/active</td>
<td>display activation status of display column &lt;primalgap&gt; (0: off, 1: auto, 2: on)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.91 display/pseudoobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/pseudoobj/active</td>
<td>display activation status of display column &lt;pseudoobj&gt; (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.92 display/separounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/separounds/active</td>
<td>display activation status of display column &lt;separounds&gt; (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.93 display/solfound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/solfound/active</td>
<td>display activation status of display column &lt;solfound&gt; (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.94 display/sols
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/sols/active</td>
<td>display activation status of display column <code>&lt;sols&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.95 display/strongbranchs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/strongbranchs/active</td>
<td>display activation status of display column <code>&lt;strongbranchs&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.96 display/time

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/time/active</td>
<td>display activation status of display column <code>&lt;time&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1 (2 for Windows without IDE)</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.97 display/vars

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/vars/active</td>
<td>display activation status of display column <code>&lt;vars&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.98 estimation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimation/coefmonossg</td>
<td>coefficient of 1 - SSG in monotone approximation of search completion</td>
<td>0.6333</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>estimation/coefmonoweight</td>
<td>coefficient of tree weight in monotone approximation of search completion</td>
<td>0.3667</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>estimation/completiontype</td>
<td>approximation of search tree completion: (a)uto, (g)ap, tree (w)eight, (m)onotone regression, (r)egression forest, (s)sg</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>Range: a, g, m, r, s, w</td>
<td></td>
</tr>
<tr>
<td>estimation/method</td>
<td>tree size estimation method: (c)ompletion, (e)nsemble, time series forecasts on either (g)ap, (l)eaf frequency, (o)pen nodes, tree (w)eight, (s)sg, or (t)ree profile or w(b)e</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>Range: b, c, e, g, l, o, s, t, w</td>
<td></td>
</tr>
<tr>
<td>estimation/regforestfilename</td>
<td>user regression forest in RFCSV format</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users
### 5.36 SCIP

#### 5.36.4.99 estimation/restarts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimation/restarts/countonlyleaves</td>
<td>should only leaves count for the minnodes parameter?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/hitcounterlim</td>
<td>limit on the number of successive samples to really trigger a restart</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/minnodes</td>
<td>minimum number of nodes before restart</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/restartactpricers</td>
<td>whether to apply a restart when active pricers are used</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/restartfactor</td>
<td>factor by which the estimated number of nodes should exceed the current number of nodes</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/restartlimit</td>
<td>restart limit</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/restartnonlinear</td>
<td>whether to apply a restart when nonlinear constraints are present</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>estimation/restarts/restartpolicy</td>
<td>restart policy: (a)always, (c)ompletion, (e)stimation, (n)ever</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>Range: a, c, e, n</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.100 estimation/ssg

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimation/ssg/nmaxsubtrees</td>
<td>the maximum number of individual SSG subtrees; -1: no limit</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 1073741823}</td>
<td></td>
</tr>
<tr>
<td>estimation/ssg/nminnodeslastsplit</td>
<td>minimum number of nodes to process between two consecutive SSG splits</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.101 estimation/treeprofile

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimation/treeprofile/enabled</td>
<td>should the event handler collect data?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>estimation/treeprofile/minnodesperdepth</td>
<td>minimum average number of nodes at each depth before producing estimations</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: $[1, \infty]$</td>
<td></td>
</tr>
<tr>
<td>5.36.4.102 expr/log</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr/log/minzerodistance</td>
<td>minimal distance from zero to enforce for child in bound tightening</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, 1]$</td>
<td></td>
</tr>
<tr>
<td>5.36.4.103 expr/pow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr/pow/minzerodistance</td>
<td>minimal distance from zero to enforce for child in bound tightening</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, 1]$</td>
<td></td>
</tr>
<tr>
<td>5.36.4.104 heuristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/useuctsubscip</td>
<td>should setting of common subscip parameters include the activation of the UCT node selector?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>5.36.4.105 heuristics/actconsdiving</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/freq</td>
<td>frequency for calling primal heuristic &lt;actconsdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;actconsdiving&gt;</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/lpresolvedomchgqu</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, \infty]$</td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, \infty]$</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/actconsdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/actconsdiving/priority</td>
<td>priority of heuristic</td>
<td>-1003700</td>
</tr>
</tbody>
</table>

5.36.4.106 heuristics/adaptivediving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/adaptivediving/bestsolweight</td>
<td>weight of incumbent solutions compared to other solutions in computation of LP iteration limit</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/adaptivediving/epsilon</td>
<td>parameter that increases probability of exploration among divesets (only active if sel-type is 'e')</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/adaptivediving/freq</td>
<td>frequency for calling primal heuristic &lt;adaptivediving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>5</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/adaptivediving/freqofs</td>
<td>frequency offset for calling primal heuristic</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/scoretype</td>
<td>score parameter for selection: minimize either average 'n'odes, LP 'i'terations,backtrack/'o'conflict ratio, 'd'epth, 1 / 's'olutions, or 1 / solutions'u'ccess</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/selconfidencecoeff</td>
<td>coefficient c to decrease initial confidence</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>(calls + 1.0) / (calls + c) in scores</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/seltype</td>
<td>selection strategy: (e)psilon-greedy, (w)eighted distribution, (n)ext diving</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/maxdepth</td>
<td>maximal depth level to call primal heuristic</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/priority</td>
<td>priority of heuristic</td>
<td>-70000</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/useadaptivecontext</td>
<td>should the heuristic use its own statistics, or shared statistics?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/adaptivediving/maxdepth</td>
<td>maximal depth level to call primal heuristic</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/priority</td>
<td>priority of heuristic</td>
<td>-70000</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
<tr>
<td>heuristics/adaptivediving/useadaptivecontext</td>
<td>should the heuristic use its own statistics, or shared statistics?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>&lt;adaptivediving&gt;</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.107 heuristics/alns

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/alns/freq</td>
<td>frequency for calling primal heuristic &lt;alns&gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;alns&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/nodesofs</td>
<td>offset added to the nodes budget</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/nodesquot</td>
<td>fraction of nodes compared to the main SCIP for budget computation</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/nodesquotmin</td>
<td>lower bound fraction of nodes compared to the main SCIP for budget computation</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/nsolslim</td>
<td>limit on the number of improving solutions in a sub- SCIP call</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/seed</td>
<td>initial random seed for bandit algorithms and random decisions by neighborhoods</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/alns/adjustfixingrate</td>
<td>should the heuristic adjust the target fixing rate based on the success? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/alns/adjustminimprove</td>
<td>should the factor by which the minimum improvement is bound be dynamically updated? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/alns/adjusttargetnodes</td>
<td>should the target nodes be dynamically adjusted? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/alns/alpha</td>
<td>parameter to increase the confidence width in UCB Range: [0, 100]</td>
<td>0.0016</td>
</tr>
<tr>
<td>heuristics/alns/banditalgo</td>
<td>the bandit algorithm: (u)pper confidence bounds, (e)xp.3, epsilon (g)reedy Range: u, e, g</td>
<td>u</td>
</tr>
<tr>
<td>heuristics/alns/beta</td>
<td>reward offset between 0 and 1 at every observation for Exp.3 Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/alns/copycuts</td>
<td>should cutting planes be copied to the sub-SCIP? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/alns/domorefixings</td>
<td>should the ALNS heuristic do more fixings by itself based on variable prioritization until the target fixing rate is reached? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/alns/eps</td>
<td>increase exploration in epsilon-greedy bandit algorithm Range: [0, 1]</td>
<td>0.468584</td>
</tr>
<tr>
<td>heuristics/alns/fixtol</td>
<td>tolerance by which the fixing rate may be missed without generic fixing Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/alns/gamma</td>
<td>weight between uniform (gamma ~ 1) and weight driven (gamma ~ 0) probability distribution for exp3 Range: [0, 1]</td>
<td>0.0704146</td>
</tr>
<tr>
<td>heuristics/alns/initduringroot</td>
<td>should the heuristic be executed multiple times during the root node? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/alns/maxcallssamesol</td>
<td>number of allowed executions of the heuristic on the same incumbent solution (-1: no limit, 0: number of active neighborhoods) Range: {-1, ..., 100}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/alns/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;alns&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/alns/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/alns/minimprovehigh</td>
<td>upper bound for the minimal improvement over the incumbent Range: [0, 1]</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/alns/minimprovelow</td>
<td>lower threshold for the minimal improvement over the incumbent Range: [0, 1]</td>
<td>0.01</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/alns/minnodes</td>
<td>minimum number of nodes required to start a sub-SCIP</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/priority</td>
<td>priority of heuristic &lt;alns&gt;</td>
<td>-1100500</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/resetweights</td>
<td>should the bandit algorithms be reset when a new problem is read?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/rewardbaseline</td>
<td>the reward baseline to separate successful and failed calls</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 0.99]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/rewardcontrol</td>
<td>reward control to increase the weight of the simple solution indicator and decrease the weight of the closed gap reward</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/rewardfilename</td>
<td>file name to store all rewards and the selection of the bandit</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/scalebyeffort</td>
<td>should the reward be scaled by the effort?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/startminimprove</td>
<td>initial factor by which ALNS should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/subsciprandseeds</td>
<td>should random seeds of sub-SCIPs be altered to increase diversification?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/targetnodefactor</td>
<td>factor by which target node number is eventually increased</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 100000]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/unfixtol</td>
<td>tolerance by which the fixing rate may be exceeded without generic unfixing</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/usedistances</td>
<td>distances from fixed variables be used for variable prioritization</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/uselocalredcost</td>
<td>should local reduced costs be used for generic (un)fixing?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/usepscost</td>
<td>should pseudo cost scores be used for variable prioritization</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/useredcost</td>
<td>should reduced cost scores be used for variable prioritization</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/usesubscipheurs</td>
<td>should the heuristic activate other sub-SCIP heuristics during its search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/waitingnodes</td>
<td>number of nodes since last incumbent solution that the heuristic should wait</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/crossover/active</td>
<td>is this neighborhood active?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/crossover/maxfixingrate</td>
<td>maximum fixing rate for this neighborhood</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/crossover/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/crossover/nsols</td>
<td>the number of solutions that crossover should combine</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 10}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/crossover/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.109 heuristics/alns/dins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/dins/active</td>
<td>is this neighborhood active?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/dins/maxfixingrate</td>
<td>maximum fixing rate for this neighborhood</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/dins/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/dins/npooolsols</td>
<td>number of pool solutions where binary solution values must agree</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/dins/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.110 heuristics/alns/localbranching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/localbranching/active</td>
<td>is this neighborhood active?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/localbranching/maxfixingrate</td>
<td>maximum fixing rate for this neighborhood</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/localbranching/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/localbranching/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.111 heuristics/alns/mutation


<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| heuristics/alns/mutation/active | is this neighborhood active?  
Range: boolean | 1 |
| heuristics/alns/mutation/maxfixingrate | maximum fixing rate for this neighborhood  
Range: [0, 1] | 0.9 |
| heuristics/alns/mutation/minfixingrate | minimum fixing rate for this neighborhood  
Range: [0, 1] | 0.3 |
| heuristics/alns/mutation/priority | positive call priority to initialize bandit algorithms  
Range: [0.01, 1] | 1 |

### 5.36.4.112 heuristics/alns/proximity

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| heuristics/alns/proximity/active | is this neighborhood active?  
Range: boolean | 1 |
| heuristics/alns/proximity/maxfixingrate | maximum fixing rate for this neighborhood  
Range: [0, 1] | 0.9 |
| heuristics/alns/proximity/minfixingrate | minimum fixing rate for this neighborhood  
Range: [0, 1] | 0.3 |
| heuristics/alns/proximity/priority | positive call priority to initialize bandit algorithms  
Range: [0.01, 1] | 1 |

### 5.36.4.113 heuristics/alns/rens

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| heuristics/alns/rens/active | is this neighborhood active?  
Range: boolean | 1 |
| heuristics/alns/rens/maxfixingrate | maximum fixing rate for this neighborhood  
Range: [0, 1] | 0.9 |
| heuristics/alns/rens/minfixingrate | minimum fixing rate for this neighborhood  
Range: [0, 1] | 0.3 |
| heuristics/alns/rens/priority | positive call priority to initialize bandit algorithms  
Range: [0.01, 1] | 1 |

### 5.36.4.114 heuristics/alns/rins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| heuristics/alns/rins/active | is this neighborhood active?  
Range: boolean | 1 |
| heuristics/alns/rins/maxfixingrate | maximum fixing rate for this neighborhood  
Range: [0, 1] | 0.9 |
### Option Description Default

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/alns/rins/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/rins/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.115 heuristics/alns/trustregion

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/alns/trustregion/violpenalty</td>
<td>the penalty for each change in the binary variables from the candidate solution</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/alns/trustregion/active</td>
<td>is this neighborhood active?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/trustregion/maxfixingrate</td>
<td>maximum fixing rate for this neighborhood</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/trustregion/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/trustregion/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.116 heuristics/alns/zeroobjective

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/alns/zeroobjective/active</td>
<td>is this neighborhood active?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/zeroobjective/maxfixingrate</td>
<td>maximum fixing rate for this neighborhood</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/zeroobjective/minfixingrate</td>
<td>minimum fixing rate for this neighborhood</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/alns/zeroobjective/priority</td>
<td>positive call priority to initialize bandit algorithms</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0.01, 1]</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.117 heuristics/bound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/bound/bound</td>
<td>to which bound should integer variables be fixed?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>('l'ower, 'u'pper, or 'b'oth)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: l, u, b</td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/freq</td>
<td>frequency for calling primal heuristic &lt;bound&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;bound&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;bound&gt;</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1: no limit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/maxproprounds</td>
<td>maximum number of propagation rounds during probing (-1 infinity, -2 parameter settings)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/onlywithoutsol</td>
<td>Should heuristic only be executed if no primal solution was found, yet?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/bound/priority</td>
<td>priority of heuristic &lt;bound&gt;</td>
<td>-1107000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.118  heuristics/clique

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/freq</td>
<td>frequency for calling primal heuristic &lt;clique&gt; (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;clique&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/minintfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/minmipfixingrate</td>
<td>minimum percentage of fixed variables in the sub-MIP</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/maxbacktracks</td>
<td>maximum number of backtracks during the fixing process</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;clique&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/maxproprounds</td>
<td>maximum number of propagation rounds during probing (-1 infinity)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/minimprove</td>
<td>factor by which clique heuristic should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/clique/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/priority</td>
<td>priority of heuristic &lt;clique&gt;</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/clique/uselockfixings</td>
<td>should more variables be fixed based on variable locks if the fixing rate was not reached?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.119 heuristics/coefdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/coefdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/freq</td>
<td>frequency for calling primal heuristic &lt;coefdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;coefdiving&gt;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;coefdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>where diving is performed (0.0: no limit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/coefdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.120 heuristics/completesol

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/completesol/beforepresol</td>
<td>should the heuristic run before presolving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/completesol/freq</td>
<td>frequency for calling primal heuristic &lt;completesol&gt; (-1: never, 0: only at depth freqofs) Range: [-1, ..., 65534]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/completesol/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;completesol&gt; Range: {0, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/completesol/ignorecont</td>
<td>should number of continuous variables be ignored? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/completesol/maxcontvars</td>
<td>maximal number of continuous variables after presolving Range: {-1, ..., (\infty)}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/completesol/maxlpiter</td>
<td>maximal number of LP iterations (-1: no limit) Range: {-1, ..., (\infty)}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/completesol/maxproprops</td>
<td>maximal number of iterations in propagation (-1: no limit) Range: {-1, ..., (\infty)}</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/completesol/maxunknownrate</td>
<td>maximal rate of unknown solution values Range: [0, 1]</td>
<td>0.85</td>
</tr>
<tr>
<td>heuristics/completesol/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes Range: {0, ..., (\infty)}</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/completesol/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/completesol/solutions</td>
<td>heuristic stops, if the given number of improving solutions were found (-1: no limit) Range: {-1, ..., (\infty)}</td>
<td>5</td>
</tr>
</tbody>
</table>

Options for expert users
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/completesol/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/completesol/boundwidening</td>
<td>bound widening factor applied to continuous variables (0: fix variables to given solution values, 1: relax to global bounds)</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/completesol/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/completesol/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;completesol&gt; (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/completesol/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/completesol/minimprove</td>
<td>factor by which the incumbent should be improved at least</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/completesol/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/completesol/objweight</td>
<td>weight of the original objective function (1: only original objective)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/completesol/priority</td>
<td>priority of heuristic &lt;completesol&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 5.36.4.121 heuristics/conflictdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/conflictdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/conflictdiving/freq</td>
<td>frequency for calling primal heuristic &lt;conflictdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/conflictdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;conflictdiving&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td>heuristics/conflictdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.15</td>
</tr>
<tr>
<td>heuristics/conflictdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/likecoef</td>
<td>perform rounding like coefficient diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/lockweight</td>
<td>weight used in a convex combination of conflict and variable locks</td>
<td>0.75</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;conflictdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/conflictdiving/maxviol</td>
<td>try to maximize the violation</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/conflictdiving/minconflictlocks</td>
<td>minimal number of conflict locks per variable</td>
<td>5</td>
</tr>
<tr>
<td>heuristics/conflictdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/conflictdiving/priority</td>
<td>priority of heuristic &lt;conflictdiving&gt;</td>
<td>-1000100</td>
</tr>
</tbody>
</table>

5.36.4.122 heuristics/crossover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/crossover/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/crossover/freq</td>
<td>frequency for calling primal heuristic &lt;crossover&gt; (-1: never, 0: only at depth freqofs)</td>
<td>30</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>heuristics/crossover/freqofs</td>
<td>frequency offset for calling primal heuristic</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>&lt;crossover&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed</td>
<td>0.666</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/nusedsols</td>
<td>number of solutions to be taken into account</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options for expert users</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/dontwaitatroot</td>
<td>should the nwaitingnodes parameter be ignored at the root node?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/maxdepth</td>
<td>maximal depth level to call primal heuristic</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>&lt;crossover&gt; (-1: no limit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/minimprove</td>
<td>factor by which Crossover should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/permute</td>
<td>should the subproblem be permuted to increase diversification?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/priority</td>
<td>priority of heuristic &lt;crossover&gt;</td>
<td>-1104000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/randomization</td>
<td>should the choice which sols to take be randomized?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/crossover/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.123 heuristics/dins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dins/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/freq</td>
<td>frequency for calling primal heuristic &lt;dins&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;dins&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the incumbent's neighborhood to be searched</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/solnum</td>
<td>number of pool-solutions to be checked for flag array update (for hard fixing of binary variables)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dins/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;dins&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/minimprove</td>
<td>factor by which dins should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/priority</td>
<td>priority of heuristic &lt;dins&gt;</td>
<td>-1105000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/dins/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/dins/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.124 heuristics/distributiondiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/distributiondiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/distributiondiving/freq</td>
<td>frequency for calling primal heuristic &lt;distributiondiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/distributiondiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;distributiondiving&gt;</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/distributiondiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td>heuristics/distributiondiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/distributiondiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/distributiondiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;distributiondiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxdiveavgquot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxdiveubquot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/distributiondiving/maxdiveubquotnosol</td>
<td>maximal UBUQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
</tbody>
</table>
## heuristics/distributiondiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/distributiondiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/distributiondiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/distributiondiving/priority</td>
<td>priority of heuristic &lt;distributiondiving&gt;</td>
<td>-1003300</td>
</tr>
<tr>
<td>heuristics/distributiondiving/scoreparam</td>
<td>the score: largest 'd'ifference, 'l'owest cumulative probability, 'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p., 'w'otes lowest c.p., 'r'evolving</td>
<td>r</td>
</tr>
</tbody>
</table>

## heuristics/dps

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dps/freq</td>
<td>frequency for calling primal heuristic &lt;dps&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dps/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;dps&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dps/maxiterations</td>
<td>maximal number of iterations</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/dps/maxlinkscore</td>
<td>maximal linking score of used decomposition (equivalent to percentage of linking constraints)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dps/penalty</td>
<td>multiplier for absolute increase of penalty parameters (0: no increase)</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/dps/reoptimize</td>
<td>should the problem get reoptimized with the original objective function?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dps/reuse</td>
<td>should solutions get reused in subproblems?</td>
<td>0</td>
</tr>
</tbody>
</table>

### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dps/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;dps&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dps/priority</td>
<td>priority of heuristic &lt;dps&gt;</td>
<td>75000</td>
</tr>
</tbody>
</table>

## heuristics/dualval

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dualval/dynamicdepth</td>
<td>says if and how the recursion depth is computed at runtime</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/freq</td>
<td>frequency for calling primal heuristic &lt;dualval&gt; (-1: never, 0: only at depth freqos)</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/dualval/freqofs</td>
<td>frequency offset for calling primal heuristic</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/heurverblevel</td>
<td>verblevel of the heuristic, default is 0 to display nothing</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/lambdobj</td>
<td>scaling factor for the objective function</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/lambdaslack</td>
<td>value added to objective of slack variables, must not be zero</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dualval/maxcalls</td>
<td>maximal number of recursive calls of the heuristic (if dynamicdepth is off)</td>
<td>25</td>
</tr>
<tr>
<td>heuristics/dualval/maxequalranks</td>
<td>maximal number of variables that may have maximal rank, quit if there are more, turn off by setting -1</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/dualval/mingap</td>
<td>minimal gap for which we still run the heuristic, if gap is less we return without doing anything</td>
<td>5</td>
</tr>
<tr>
<td>heuristics/dualval/nlpverblevel</td>
<td>verblevel of the nlp solver, can be 0 or 1</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/onlyleaves</td>
<td>disable the heuristic if it was not called at a leaf of the B&amp;B tree</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/rankvalue</td>
<td>number of ranks that should be displayed when the heuristic is called</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/dualval/relaxcontvars</td>
<td>relax the continuous variables</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/relaxindicators</td>
<td>relax the indicator variables by introducing continuous copies</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/dualval/forceimprovements</td>
<td>exit if objective doesn’t improve</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/maxdepth</td>
<td>maximal depth level to call primal heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dualval/onlycheaper</td>
<td>add constraint to ensure that discrete vars are improving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dualval/priority</td>
<td>priority of heuristic</td>
<td>-10</td>
</tr>
</tbody>
</table>

5.36.4.127  heuristics/farkasdiving
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/farkasdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/freq</td>
<td>frequency for calling primal heuristic &lt;farkasdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;farkasdiving&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/checkcands</td>
<td>should diving candidates be checked before running?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;farkasdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxdiveubquotnosol</td>
<td>maximal UBOUND when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/farkasdiving/maxobjocc</td>
<td>maximal occurrence factor of an objective coefficient</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/farkasdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/farkasdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/farkasdiving/objdynamism</td>
<td>minimal objective dynamism (log) to run</td>
<td>0.0001</td>
</tr>
<tr>
<td>heuristics/farkasdiving/priority</td>
<td>priority of heuristic &lt;farkasdiving&gt;</td>
<td>-900000</td>
</tr>
<tr>
<td>heuristics/farkasdiving/rootsuccess</td>
<td>should the heuristic only run within the tree if at least one solution was found at the root node?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/farkasdiving/scalescore</td>
<td>should the score be scaled?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/farkasdiving/scaletype</td>
<td>scale score by [f]ractionality or [i]mpact on farkasproof</td>
<td>i</td>
</tr>
</tbody>
</table>

### 5.36.4.128 heuristics/feaspump

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/feaspump/alpha</td>
<td>initial weight of the objective function in the convex combination</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/alphadiff</td>
<td>threshold difference for the convex parameter to perform perturbation</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/beforecuts</td>
<td>should the feasibility pump be called at root node before cut separation?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/freq</td>
<td>frequency for calling primal heuristic &lt;feaspump&gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/feaspump/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;feaspump&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/feaspump/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/feaspump/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/feaspump/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the neighborhood to be searched in stage 3</td>
<td>18</td>
</tr>
<tr>
<td>heuristics/feaspump/objfactor</td>
<td>factor by which the regard of the objective is decreased in each round, 1.0 for dynamic</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/feaspump/pertsolfound</td>
<td>should a random perturbation be performed if a feasible solution was found?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/feaspump/stage3</td>
<td>should we solve a local branching sub-MIP if no solution could be found? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/feaspump/usefp20</td>
<td>should an iterative round-and-propagate scheme be used to find the integral points? Range: boolean</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/feaspump/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/cyclelength</td>
<td>maximum length of cycles to be checked explicitly in each round Range: {1, ..., 100}</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/feaspump/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;feaspump&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/feaspump/maxloops</td>
<td>maximal number of pumping loops (-1: no limit) Range: {-1, ..., ∞}</td>
<td>10000</td>
</tr>
<tr>
<td>heuristics/feaspump/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/feaspump/maxstallloops</td>
<td>maximal number of pumping rounds without fractionality improvement (-1: no limit) Range: {-1, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/feaspump/minflips</td>
<td>minimum number of random variables to flip, if a 1-cycle is encountered Range: {1, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/feaspump/perturbfreq</td>
<td>number of iterations until a random perturbation is forced Range: {1, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/feaspump/priority</td>
<td>priority of heuristic &lt;feaspump&gt; Range: {-536870912, ..., 536870911}</td>
<td>-100000</td>
</tr>
</tbody>
</table>

5.36.4.129 heuristics/fixandinfer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fixandinfer/freq</td>
<td>frequency for calling primal heuristic &lt;fixandinfer&gt; (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/fixandinfer/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;fixandinfer&gt; Range: {0, ..., 65534}</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fixandinfer/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;fixandinfer&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/fixandinfer/minfixings</td>
<td>minimal number of fixings to apply before dive may be aborted Range: {0, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/fixandinfer/priority</td>
<td>priority of heuristic &lt;fixandinfer&gt; Range: {-536870912, ..., 536870911}</td>
<td>-500000</td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fixandinfer/proprounds</td>
<td>maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.130 heuristics/fracdving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fracdving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/freq</td>
<td>frequency for calling primal heuristic &lt;fracdving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;fracdving&gt;</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/only4pbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fracdving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;fracdving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fracdving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>
### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/gins/consecutiveblocks</td>
<td>should blocks be treated consecutively (sorted by ascending label?)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/gins/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/gins/fixedcontvars</td>
<td>should continuous variables outside the neighborhoods be fixed?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/gins/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;gins&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/gins/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/gins/minimprove</td>
<td>factor by which gins should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>heuristics/gins/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/gins/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/gins/overlap</td>
<td>overlap of blocks between runs - 0.0: no overlap, 1.0: shift by only 1 block</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/gins/potential</td>
<td>the reference point to compute the neighborhood potential: (r)oot, (l)ocallp, or (p)seudo solution</td>
<td>r</td>
</tr>
<tr>
<td>heuristics/gins/priority</td>
<td>priority of heuristic &lt;gins&gt;</td>
<td>-1103000</td>
</tr>
<tr>
<td>heuristics/gins/relaxdenseconss</td>
<td>should dense constraints (at least as dense as 1 - minfixingrate) be ignored by connectivity graph?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/gins/rollhorizonlimfac</td>
<td>limiting percentage for variables already used in sub-SCIPs to terminate rolling horizon approach</td>
<td>0.4</td>
</tr>
<tr>
<td>heuristics/gins/usedecomp</td>
<td>should user decompositions be considered, if available?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/gins/usedecomrollhorizon</td>
<td>should user decompositions be considered for initial selection in rolling horizon, if available?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/gins/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/gins/userollinghorizon</td>
<td>should the heuristic solve a sequence of sub-MIP's around the first selected variable</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/gins/uselselfallback</td>
<td>should random initial variable selection be used if decomposition was not successful?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.132 heuristics/guideddiving

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/guideddiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/guideddiving/freq</td>
<td>frequency for calling primal heuristic &lt;guideddiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/guideddiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;guideddiving&gt;</td>
<td>7</td>
</tr>
<tr>
<td>heuristics/guideddiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/guideddiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., \infty}</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations Range: {0, ..., \infty}</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations Range: [0, \infty]</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/guideddiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;\text{guideddiving}&gt;) (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, \infty]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, \infty]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/guideddiving/minreldepth</td>
<td>minimal relative depth to start diving Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/guideddiving/priority</td>
<td>priority of heuristic (&lt;\text{guideddiving}&gt;) Range: {-536870912, ..., 536870911}</td>
<td>-1007000</td>
</tr>
</tbody>
</table>

5.36.4.133 heuristics/indicator

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/indicator/freq</td>
<td>frequency for calling primal heuristic (&lt;\text{indicator}&gt;) (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/indicator/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;\text{indicator}&gt;) Range: {0, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/indicator/improvesols</td>
<td>Try to improve other solutions by one-opt? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/indicator/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;indicator&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/indicator/oneopt</td>
<td>whether the one-opt heuristic should be started</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/indicator/priority</td>
<td>priority of heuristic &lt;indicator&gt;</td>
<td>-20200</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.134 heuristics/intdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/intdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/freq</td>
<td>frequency for calling primal heuristic &lt;intdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;intdiving&gt;</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;intdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/intdiving/priority</td>
<td>priority of heuristic &lt;intdiving&gt;</td>
<td>-1003500</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.135 heuristics/intshifting

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/intshifting/freq</td>
<td>frequency for calling primal heuristic &lt;intshifting&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intshifting/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;intshifting&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/intshifting/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;intshifting&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/intshifting/priority</td>
<td>priority of heuristic &lt;intshifting&gt;</td>
<td>-10000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.136 heuristics/linesearchdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/linesearchdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/freq</td>
<td>frequency for calling primal heuristic &lt;linesearchdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;linesearchdiving&gt;</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/lpresolvedomechchgquote</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>heuristics/linesearchdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/linesearchdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;linesearchdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveavgquot</td>
<td>maximal quotient (c$lowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveubquot</td>
<td>maximal quotient (c$lowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/priority</td>
<td>priority of heuristic $linesearchdiving$</td>
<td>-1006000</td>
</tr>
</tbody>
</table>

5.36.4.137 heuristics/localbranching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/localbranching/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/localbranching/freq</td>
<td>frequency for calling primal heuristic $&lt;localbranching&gt;$ (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/localbranching/freqofs</td>
<td>frequency offset for calling primal heuristic $&lt;localbranching&gt;$</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/localbranching/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the incumbent's neighborhood to be searched</td>
<td>18</td>
</tr>
<tr>
<td>heuristics/localbranching/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/localbranching/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/localbranching/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/localbranching/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>1.5</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| heuristics/localbranching/maxdepth | maximal depth level to call primal heuristic  
<localbranching> (-1: no limit)  
Range: {-1, ..., 65534} | -1      |
| heuristics/localbranching/maxnodes | maximum number of nodes to regard in the subproblem  
Range: {0, ..., ∞} | 10000   |
| heuristics/localbranching/minimprove | factor by which localbranching should at least improve the incumbent  
Range: [0, 1] | 0.01    |
| heuristics/localbranching/minnodes | minimum number of nodes required to start the subproblem  
Range: {0, ..., ∞} | 1000    |
| heuristics/localbranching/nwaitingnodes | number of nodes without incumbent change that heuristic should wait  
Range: {0, ..., ∞} | 200     |
| heuristics/localbranching/priority | priority of heuristic <localbranching>  
Range: {-536870912, ..., 536870911} | -1102000 |
| heuristics/localbranching/uselprows | should subproblem be created out of the rows in the LP rows?  
Range: boolean | 0       |

5.36.4.138  heuristics/locks

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| heuristics/locks/freq | frequency for calling primal heuristic <locks> (-1: never, 0: only at depth freqofs)  
Range: {-1, ..., 65534} | 0       |
| heuristics/locks/freqofs | frequency offset for calling primal heuristic <locks>  
Range: {0, ..., 65534} | 0       |
| heuristics/locks/minfixingrate | minimum percentage of integer variables that have to be fixable  
Range: [0, 1] | 0.65    |
| heuristics/locks/nodesofs | number of nodes added to the contingent of the total nodes  
Range: {0, ..., ∞} | 500     |
| heuristics/locks/nodesquot | contingent of sub problem nodes in relation to the number of nodes of the original problem  
Range: [0, 1] | 0.1     |
| heuristics/locks/roundupprobability | probability for rounding a variable up in case of ties  
Range: [0, 1] | 0.67    |

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| heuristics/locks/copycuts | should all active cuts from cutpool be copied to constraints in subproblem?  
Range: boolean | 1       |
| heuristics/locks/maxdepth | maximal depth level to call primal heuristic <locks> (-1: no limit)  
Range: {-1, ..., 65534} | -1      |
| heuristics/locks/maxnodes | maximum number of nodes to regard in the subproblem  
Range: {0, ..., ∞} | 5000    |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/locks/maxproprounds</td>
<td>maximum number of propagation rounds to be performed in each propagation call (-1: no limit, -2: parameter settings)</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/locks/minfixingratelp</td>
<td>minimum fixing rate over all variables (including continuous) to solve LP</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/locks/minimprove</td>
<td>factor by which locks heuristic should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/locks/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/locks/priority</td>
<td>priority of heuristic &lt;locks&gt;</td>
<td>3000</td>
</tr>
<tr>
<td>heuristics/locks/updatelocks</td>
<td>should the locks be updated based on LP rows?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/locks/usefinalsubmit</td>
<td>should a final sub-MIP be solved to construct a feasible solution if the LP was not roundable?</td>
<td>1</td>
</tr>
</tbody>
</table>

**5.36.4.139 heuristics/lpface**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/lpface/freq</td>
<td>frequency for calling primal heuristic &lt;lpface&gt; (-1: never, 0: only at depth freqofs)</td>
<td>15</td>
</tr>
<tr>
<td>heuristics/lpface/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;lpface&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/lpface/minfixingrate</td>
<td>required percentage of fixed integer variables in sub-MIP to run</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/lpface/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/lpface/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Options for expert users

- heuristics/lpface/copycuts: if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean
- heuristics/lpface/dualbasisequations: should dually nonbasic rows be turned into equations? Range: boolean
- heuristics/lpface/keepsbscip: should the heuristic continue solving the same sub-SCIP? Range: boolean
- heuristics/lpface/lplimfac: factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]
### heuristics/lpface

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/lpface/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;lpface&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/minpathlen</td>
<td>the minimum active search tree path length along which lower bound hasn't changed before heuristic becomes active</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65531}</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/priority</td>
<td>priority of heuristic <code>&lt;lpface&gt;</code></td>
<td>-1104010</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/subscipobjective</td>
<td>objective function in the sub-SCIP: (z)ero, (r)oot-LP-difference, (i)nference, LP (f)ractionality, (o)riginal</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>Range: f, o, r, z, i</td>
<td></td>
</tr>
<tr>
<td>heuristics/lpface/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### heuristics/mpec

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/mpec/freq</td>
<td>frequency for calling primal heuristic <code>&lt;mpec&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;mpec&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/inittheta</td>
<td>initial regularization right-hand side value</td>
<td>0.125</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 0.25]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/maxiter</td>
<td>maximum number of iterations of the MPEC loop</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/maxnlpcost</td>
<td>maximum cost available for solving NLPs per call of the heuristic</td>
<td>1e+08</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/maxnlpiter</td>
<td>maximum number of NLP iterations per solve</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/maxnunsucc</td>
<td>maximum number of consecutive calls for which the heuristic did not find an improving solution</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/mingapleft</td>
<td>minimum amount of gap left in order to call the heuristic</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/minimprove</td>
<td>factor by which heuristic should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/sigma</td>
<td>regularization update factor</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/mpec/subnlptrigger</td>
<td>maximum number of NLP iterations per solve Range: [0, 1]</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/mpec/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;mpec&gt; (&lt;-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/mpec/priority</td>
<td>priority of heuristic &lt;mpec&gt; Range: {-536870912, ..., 536870911}</td>
<td>-2050000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/multistart/freq</td>
<td>frequency for calling primal heuristic &lt;multistart&gt; (&lt;-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/multistart/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;multistart&gt; Range: {0, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/multistart/gradlimit</td>
<td>limit for gradient computations for all improve-Point() calls (0 for no limit) Range: [0, ∞]</td>
<td>5e+06</td>
</tr>
<tr>
<td>heuristics/multistart/maxboundsize</td>
<td>maximum variable domain size for unbounded variables Range: [0, ∞]</td>
<td>20000</td>
</tr>
<tr>
<td>heuristics/multistart/maxiter</td>
<td>number of iterations to reduce the maximum violation of a point Range: {0, ..., ∞}</td>
<td>300</td>
</tr>
<tr>
<td>heuristics/multistart/maxncluster</td>
<td>maximum number of considered clusters per heuristic call Range: {0, ..., ∞}</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/multistart/maxreldist</td>
<td>maximum distance between two points in the same cluster Range: [0, ∞]</td>
<td>0.15</td>
</tr>
<tr>
<td>heuristics/multistart/minimprfac</td>
<td>minimum required improving factor to proceed in improvement of a single point Range: real</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/multistart/minimpriter</td>
<td>number of iteration when checking the minimum improvement Range: {1, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/multistart/nrndpoints</td>
<td>number of random points generated per execution call Range: {0, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/multistart/onlynlps</td>
<td>should the heuristic run only on continuous problems? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/multistart/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;multistart&gt; (&lt;-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/multistart/priority</td>
<td>priority of heuristic &lt;multistart&gt; Range: {-536870912, ..., 536870911}</td>
<td>-2100000</td>
</tr>
</tbody>
</table>
### 5.36.4.142 heuristics/mutation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/mutation/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/freq</td>
<td>frequency for calling primal heuristic &lt;mutation&gt;</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>((-1: \text{never}, 0: \text{only at depth freqofs}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;mutation&gt;</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/minfixingrate</td>
<td>percentage of integer variables that have to be fixed</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-06, 0.999999]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;mutation&gt;</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>((-1: \text{no limit}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/minimprove</td>
<td>factor by which mutation should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/priority</td>
<td>priority of heuristic &lt;mutation&gt;</td>
<td>-1103010</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/mutation/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.143 heuristics/nlpdiving
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/nlpdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/fixquot</td>
<td>percentage of fractional variables that should be fixed before the next NLP solve</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/freq</td>
<td>frequency for calling primal heuristic &lt;nlpdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;nlpdiving&gt;</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxfeasnlps</td>
<td>maximal number of NLPs with feasible solution to solve during one dive</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxnlpiterabs</td>
<td>minimal absolute number of allowed NLP iterations</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxnlpiterrel</td>
<td>additional allowed number of NLP iterations relative to successfully found solutions</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/minsuccquot</td>
<td>heuristic will not run if less then this percentage of calls succeeded (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/nlpfastfail</td>
<td>should the NLP solver stop early if it converges slow?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/prefercov</td>
<td>should variables in a minimal cover be preferred?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/solvesubmip</td>
<td>should a sub-MIP be solved if all cover variables are fixed?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/varselrule</td>
<td>which variable selection should be used? ('f'ractionality, 'c'oefficient, 'p'suedocost, 'g'uided, 'd'double, 'v'eclen)</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>Range: f, c, p, g, d, v</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/nlpdiving/lp</td>
<td>should the LP relaxation be solved before the NLP relaxation?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;nlpdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveavgquot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>
### Option Description Default

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/nlpdiving/maxdiveubquot</td>
<td>maximal quotient (cutoffbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/minreldepth</td>
<td>minimal relative depth to start diving Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/nlpdiving/nlpstart</td>
<td>which point should be used as starting point for the NLP solver? ('n'one, last 'f'easible, from dive's'tart) Range: f, n, s</td>
<td>8</td>
</tr>
<tr>
<td>heuristics/nlpdiving/preferlpfracs</td>
<td>prefer variables that are also fractional in LP solution? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/nlpdiving/priority</td>
<td>priority of heuristic &lt;nlpdiving&gt; Range: {-536870912, ..., 536870911}</td>
<td>-1003010</td>
</tr>
</tbody>
</table>

### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/nlpdiving/depthfac</td>
<td>maximal diving depth: number of binary/integer variables times depthfac Range: [0, ∞]</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/nlpdiving/depthfacnosol</td>
<td>maximal diving depth factor if no feasible solution was found yet Range: [0, ∞]</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;nlpdiving&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.36.4.144 heuristics/objpscostdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/objpscostdiving/freq</td>
<td>frequency for calling primal heuristic &lt;objpscostdiving&gt; (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;objpscostdiving&gt; Range: {0, ..., 65534}</td>
<td>4</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations Range: {0, ..., ∞}</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to total iteration number Range: [0, 1]</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/objpscostdiving/depthfac</td>
<td>maximal diving depth: number of binary/integer variables times depthfac Range: [0, ∞]</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/depthfacnosol</td>
<td>maximal diving depth factor if no feasible solution was found yet Range: [0, ∞]</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;objpscostdiving&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
</tbody>
</table>
### 5.36.4.145 heuristics/octane

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/octane/freq</td>
<td>frequency for calling primal heuristic <code>&lt;octane&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/octane/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;octane&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/octane/ffirst</td>
<td>number of 0-1-points to be tested at first whether they violate a common row</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/octane/fmax</td>
<td>number of 0-1-points to be tested as possible solutions by OCTANE</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/octane/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;octane&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/octane/priority</td>
<td>priority of heuristic <code>&lt;octane&gt;</code></td>
<td>-1008000</td>
</tr>
<tr>
<td>heuristics/octane/useavgubray</td>
<td>should the weighted average of the nonbasic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useavggray</td>
<td>should the average of the basic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useavgwgtray</td>
<td>should the weighted average of the basic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/usediffray</td>
<td>should the difference between the root solution and the current LP solution be used as one ray direction?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/octane/usefracspace</td>
<td>execute OCTANE only in the space of fractional variables (TRUE) or in the full space?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useobjray</td>
<td>should the inner normal of the objective be used as one ray direction?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.146 heuristics/ofins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/ofins/freq</td>
<td>frequency for calling primal heuristic <code>&lt;ofins&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/ofins/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;ofins&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/maxchange</td>
<td>maximal rate of change per coefficient to get fixed</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/maxchangerate</td>
<td>maximal rate of changed coefficients</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/ofins/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;ofins&gt;</code> (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/minimprove</td>
<td>factor by which RENS should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/ofins/priority</td>
<td>priority of heuristic <code>&lt;ofins&gt;</code></td>
<td>60000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.147 **heuristics/oneopt**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/oneopt/freq</td>
<td>frequency for calling primal heuristic <code>&lt;oneopt&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/oneopt/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;oneopt&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/oneopt/beforepresol</td>
<td>should the heuristic be called before presolving?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/oneopt/duringroot</td>
<td>should the heuristic be called before and during the root node?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/oneopt/forcelpconstruction</td>
<td>should the construction of the LP be forced even if LP solving is deactivated?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/oneopt/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;oneopt&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/oneopt/priority</td>
<td>priority of heuristic <code>&lt;oneopt&gt;</code></td>
<td>-20000</td>
</tr>
<tr>
<td>heuristics/oneopt/useloop</td>
<td>should the heuristic continue to run as long as improvements are found?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/oneopt/weightedobj</td>
<td>should the objective be weighted with the potential shifting value when sorting the shifting candidates?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.148 heuristics/padm

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/padm/assignlinking</td>
<td>should linking constraints be assigned?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/padm/freq</td>
<td>frequency for calling primal heuristic <code>&lt;padm&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/padm/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;padm&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/padm/reoptimize</td>
<td>should the problem get reoptimized with the original objective function?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/padm/timing</td>
<td>should the heuristic run before or after the processing of the node? (0: before, 1: after, 2: both)</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/padm/admiterations</td>
<td>maximal number of ADM iterations in each penalty loop</td>
<td>4</td>
</tr>
<tr>
<td>heuristics/padm/gap</td>
<td>mipgap at start</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/padm/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;padm&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/padm/maxnodes</td>
<td>maximum number of nodes to regard in all subproblems</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/padm/minnodes</td>
<td>minimum number of nodes to regard in one subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/padm/nodefac</td>
<td>factor to control nodelimits of subproblems</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/padm/original</td>
<td>should the original problem be used? This is only for testing and not recommended!</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/padm/penaltyiterations</td>
<td>maximal number of penalty iterations</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100000}</td>
<td></td>
</tr>
<tr>
<td>heuristics/padm/priority</td>
<td>priority of heuristic &lt;padm&gt;</td>
<td>70000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/padm/scaling</td>
<td>enable sigmoid rescaling of penalty parameters</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.149 heuristics/proximity

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/proximity/freq</td>
<td>frequency for calling primal heuristic &lt;proximity&gt;</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1: never, 0: only at depth freqofs)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;proximity&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/binvarquot</td>
<td>threshold for percentage of binary variables required to start</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, 1}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/lpitersquot</td>
<td>quotient of sub-MIP LP iterations with respect to LP iterations so far</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, 1}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;proximity&gt;</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1: no limit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/maxlpiters</td>
<td>maximum number of LP iterations to be performed in the subproblem</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/mingap</td>
<td>minimum primal-dual gap for which the heuristic is executed</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/minimprove</td>
<td>factor by which proximity should at least improve the incumbent</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>Range: {0, 1}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/minlpiters</td>
<td>minimum number of LP iterations performed in subproblem</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/nodesquot</td>
<td>sub-MIP node limit w.r.t number of original nodes</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/priority</td>
<td>priority of heuristic &lt;proximity&gt;</td>
<td>-2000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/proximity/restart</td>
<td>should the heuristic immediately run again on its newly found solution?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/usefinallp</td>
<td>should the heuristic solve a final LP in case of continuous objective variables?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/uselprows</td>
<td>should subproblem be constructed based on LP row information?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/proximity/waitingnodes</td>
<td>waiting nodes since last incumbent before heuristic is executed</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: ( {0, \ldots, \infty} )</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.150 heuristics/pscostdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/pscostdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/freq</td>
<td>frequency for calling primal heuristic (&lt;\text{pscostdiving}&gt;) (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, \ldots, 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;\text{pscostdiving}&gt;)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \ldots, 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \ldots, \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \ldots, \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/pscostdiving/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/pscostdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;\text{pscostdiving}&gt;) (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, \ldots, 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/pscstdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/pscstdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/pscstdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/pscstdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/pscstdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/pscstdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/pscstdiving/priority</td>
<td>priority of heuristic &lt;pscstdiving&gt;</td>
<td>-100200</td>
</tr>
</tbody>
</table>

5.36.4.151 heuristics/randrounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/randrounding/freq</td>
<td>frequency for calling primal heuristic &lt;randrounding&gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/randrounding/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;randrounding&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/randrounding/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;randrounding&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/randrounding/maxproprounds</td>
<td>limit of rounds for each propagation call</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/randrounding/oncepernode</td>
<td>should the heuristic only be called once per node?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/randrounding/priority</td>
<td>priority of heuristic &lt;randrounding&gt;</td>
<td>-200</td>
</tr>
<tr>
<td>heuristics/randrounding/propagateonlyroot</td>
<td>should the probing part of the heuristic be applied exclusively at the root node?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/randrounding/usesimplerounding</td>
<td>should the heuristic apply the variable lock strategy of simple rounding, if possible?</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.152 heuristics/rens
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rens/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/rens/freq</td>
<td>frequency for calling primal heuristic (&lt;\text{rens}&gt;) (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;\text{rens}&gt;)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/startsol</td>
<td>solution that is used for fixing values ('lp relaxation, 'nlp relaxation)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: n, l</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/binarybounds</td>
<td>should general integers get binary bounds ([\text{floor}.()],\text{ceil}.()]) ?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/copycuts</td>
<td>if (\text{useprows} == \text{FALSE}), should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/extratime</td>
<td>should the RENS sub-CIP get its own full time limit? This is only for testing and not recommended!</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/fullscale</td>
<td>should the RENS sub-CIP be solved with cuts, conflicts, strong branching,... This is only for testing and not recommended!</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, \infty]</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;\text{rens}&gt;) (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/minimprove</td>
<td>factor by which RENS should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>heuristics/rens/priority</td>
<td>priority of heuristic &lt;rens&gt;</td>
<td>-1100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.153 heuristics/re.optsols

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/re.optsols/freq</td>
<td>frequency for calling primal heuristic &lt;re.optsols&gt; (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/re.optsols/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;re.optsols&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/re.optsols/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;re.optsols&gt; (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/re.optsols/maxruns</td>
<td>check solutions of the last k runs. (-1: all)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/re.optsols/maxsols</td>
<td>maximal number solutions which should be checked. (-1: all)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/re.optsols/priority</td>
<td>priority of heuristic &lt;re.optsols&gt;</td>
<td>40000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.154 heuristics/repair

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/repair/filename</td>
<td>file name of a solution to be used as infeasible starting</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>point, [-] if not available</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/freq</td>
<td>frequency for calling primal heuristic &lt;repair&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;repair&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/repair/roundit</td>
<td>True : fractional variables which are not fractional in the given solution are rounded, FALSE : solving process of this heuristic is stopped. Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/repair/useobjfactor</td>
<td>should a scaled objective function for original variables be used in repair subproblem? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/repair/useslackvars</td>
<td>should slack variables be used in repair subproblem? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/repair/usevarfix</td>
<td>should variable fixings be used in repair subproblem? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Options for expert users</strong></td>
<td></td>
</tr>
<tr>
<td>heuristics/repair/alpha</td>
<td>factor for the potential of var fixings Range: [0, 100]</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/repair/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;repair&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/repair/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/repair/minnodes</td>
<td>minimum number of nodes required to start the subproblem Range: {0, ..., ∞}</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/repair/priority</td>
<td>priority of heuristic &lt;repair&gt; Range: {-536870912, ..., 536870911}</td>
<td>-20</td>
</tr>
</tbody>
</table>

### 5.36.4.155 heuristics/rins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rins/freq</td>
<td>frequency for calling primal heuristic &lt;rins&gt; (-1: never, 0: only at depth freqofs) Range: [-1, ..., 65534]</td>
<td>25</td>
</tr>
<tr>
<td>heuristics/rins/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;rins&gt; Range: {0, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rins/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed Range: [0, 1]</td>
<td>0.3</td>
</tr>
<tr>
<td>heuristics/rins/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/rins/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td><strong>Options for expert users</strong></td>
<td></td>
</tr>
<tr>
<td>heuristics/rins/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rins/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/rins/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;rins&gt; (-1: no limit) Range: {-1, ..., 65534}</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/rins/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/rins/minimprove</td>
<td>factor by which rins should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/rins/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/rins/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/rins/priority</td>
<td>priority of heuristic <code>&lt;rins&gt;</code></td>
<td>-1101000</td>
</tr>
<tr>
<td>heuristics/rins/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rins/useuct</td>
<td>should uct node selection be used at the beginning of the search?</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.156  heuristics/rootsoldiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rootsoldiving/freq</td>
<td>frequency for calling primal heuristic <code>&lt;rootsoldiving&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;rootsoldiving&gt;</code></td>
<td>5</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rootsoldiving/alpha</td>
<td>soft rounding factor to fade out objective coefficients</td>
<td>0.9</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/depthfac</td>
<td>maximal diving depth: number of binary/integer variables times depthfac</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/depthfacnosol</td>
<td>maximal diving depth factor if no feasible solution was found yet</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;rootsoldiving&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit)</td>
<td>-1</td>
</tr>
</tbody>
</table>
### 5.36.4.157 heuristics/rounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rounding/freq</td>
<td>frequency for calling primal heuristic &lt;rounding&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rounding/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;rounding&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rounding/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;rounding&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/rounding/oncepernode</td>
<td>should the heuristic only be called once per node?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rounding/priority</td>
<td>priority of heuristic &lt;rounding&gt;</td>
<td>-1000</td>
</tr>
<tr>
<td>heuristics/rounding/successfactor</td>
<td>number of calls per found solution that are considered as standard success, a higher factor causes the heuristic to be called more often</td>
<td>100</td>
</tr>
</tbody>
</table>

### 5.36.4.158 heuristics/shiftandpropagate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/shiftandpropagate/freq</td>
<td>frequency for calling primal heuristic &lt;shiftandpropagate&gt; (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;shiftandpropagate&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/shiftandpropagate/binlocksfirst</td>
<td>should binary variables with no locks be preferred in the ordering?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/collectstats</td>
<td>should variable statistics be collected during probing?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/cutoffbreaker</td>
<td>The number of cutoffs before heuristic stops</td>
<td>15</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/fixbinlocks</td>
<td>should binary variables with no locks in one direction be fixed to that direction?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/impliscontinuous</td>
<td>should implicit integer variables be treated as continuous variables?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/maxcutoffquot</td>
<td>maximum percentage of allowed cutoffs before stopping the heuristic</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;shiftandpropagate&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/minfixingratelp</td>
<td>minimum fixing rate over all variables (including continuous) to solve LP</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/normalize</td>
<td>should coefficients and left/right hand sides be normalized by max row coeff?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/nozerofixing</td>
<td>should variables with a zero shifting value be delayed instead of being fixed?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/nproprounds</td>
<td>The number of propagation rounds used for each propagation</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/onlywithoutsol</td>
<td>Should heuristic only be executed if no primal solution was found, yet?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/preferbinaries</td>
<td>Should binary variables be shifted first?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/priority</td>
<td>priority of heuristic <code>&lt;shiftandpropagate&gt;</code></td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/probing</td>
<td>Should domains be reduced by probing?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/relax</td>
<td>Should continuous variables be relaxed?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/selectbest</td>
<td>should the heuristic choose the best candidate in every round? (set to FALSE for static order)?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/sortkey</td>
<td>the key for variable sorting: (n)orms down, norms (u)p, (v)iolations down, viola(t)ions up, or (r)andom</td>
<td>v</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/sortvars</td>
<td>Should variables be sorted for the heuristic?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/stopafterfeasible</td>
<td>Should the heuristic stop calculating optimal shift values when no more rows are violated?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/shiftandpropagate/updateweights</td>
<td>should row weight be increased every time the row is violated?</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.159 heuristics/shifting
## 5.36.4.160 heuristics/simplerounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/simplerounding/freq</td>
<td>frequency for calling primal heuristic &lt;simplerounding&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/simplerounding/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;simplerounding&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/simplerounding/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;simplerounding&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/simplerounding/priority</td>
<td>priority of heuristic &lt;simplerounding&gt;</td>
<td>-30</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

## 5.36.4.161 heuristics/subnlp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/subnlp/expectinfeas</td>
<td>percentage of NLP solves with infeasible status required to tell NLP solver to expect an infeasible NLP</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/feastolfactor</td>
<td>factor on SCIP feasibility tolerance for NLP solves if resolving when NLP solution not feasible in CIP</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/forbidfixings</td>
<td>whether to add constraints that forbid specific fixings that turned out to be infeasible</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/freq</td>
<td>frequency for calling primal heuristic &lt;subnlp&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/subnlp/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;subnlp&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/iterinit</td>
<td>number of iterations used for initial NLP solves</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/itermin</td>
<td>minimal number of iterations for NLP solves</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/maxpresolverounds</td>
<td>limit on number of presolve rounds in sub-SCIP</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1 for unlimited, 0 for no presolve)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: <code>{-1, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/ninitsolves</td>
<td>number of successful NLP solves until switching</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>to iterlimit guess and using success rate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/nlpverblevel</td>
<td>verbosity level of NLP solver</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65535}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/nodesfactor</td>
<td>factor on number of nodes in SCIP (plus nodesoffset) to compute itercontingent (higher value runs heuristics more frequently)</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: <code>[0, ∞]</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/nodesoffset</td>
<td>number of nodes added to the current number of nodes when computing itercontingent (higher value runs heuristic more often in early search)</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/presolveemphasis</td>
<td>presolve emphasis in sub-SCIP (0: default, 1: aggressive, 2: fast, 3: off)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 3}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/setcutoff</td>
<td>whether to set cutoff in sub-SCIP to current primal bound</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/successrateexp</td>
<td>exponent for power of success rate to be multiplied with itercontingent (lower value decreases impact of success rate)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: <code>[0, ∞]</code></td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/subnlp/keepcopy</td>
<td>whether to keep SCIP copy or to create new copy each time heuristic is applied</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;subnlp&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/opttol</td>
<td>absolute optimality tolerance to use for NLP solves</td>
<td>1e-07</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, 1}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/priority</td>
<td>priority of heuristic <code>&lt;subnlp&gt;</code></td>
<td>-2000010</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{-536870912, ..., 536870911}</code></td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.162 heuristics/trivial

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivial/freq</td>
<td>frequency for calling primal heuristic <code>&lt;trivial&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivial/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;trivial&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivial/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;trivial&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trivial/priority</td>
<td>priority of heuristic <code>&lt;trivial&gt;</code></td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{536870912, ..., 536870911}</code></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.163 heuristics/trivialnegation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivialnegation/freq</td>
<td>frequency for calling primal heuristic <code>&lt;trivialnegation&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trivialnegation/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;trivialnegation&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivialnegation/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;trivialnegation&gt;</code> (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trivialnegation/priority</td>
<td>priority of heuristic <code>&lt;trivialnegation&gt;</code></td>
<td>39990</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{536870912, ..., 536870911}</code></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.164 heuristics/trustregion

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trustregion/bestsollimit</td>
<td>limit on number of improving incumbent solutions in sub-CIP</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trustregion/freq</td>
<td>frequency for calling primal heuristic <code>&lt;trustregion&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trustregion/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;trustregion&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., 65534}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trustregion/minbinvars</td>
<td>the number of binary variables necessary to run</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>the heuristic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: <code>{1, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trustregion/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, ..., ∞}</code></td>
<td></td>
</tr>
<tr>
<td>heuristics/trustregion/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: <code>{0, 1}</code></td>
<td></td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>heuristics/trustregion/objminimprove</td>
<td>the minimum absolute improvement in the objective function value</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/trustregion/violpenalty</td>
<td>the penalty for each change in the binary variables from the candidate solution</td>
<td>100</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trustregion/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/trustregion/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>1.5</td>
</tr>
<tr>
<td>heuristics/trustregion/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;trustregion&gt; {-1: no limit}</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/trustregion/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>10000</td>
</tr>
<tr>
<td>heuristics/trustregion/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/trustregion/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/trustregion/priority</td>
<td>priority of heuristic &lt;trustregion&gt;</td>
<td>-1102010</td>
</tr>
<tr>
<td>heuristics/trustregion/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
</tbody>
</table>

---

### 5.36.4.165 heuristics/trysoI

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trysoI/freq</td>
<td>frequency for calling primal heuristic &lt;trysoI&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/trysoI/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;trysoI&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trysoI/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;trysoI&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/trysoI/priority</td>
<td>priority of heuristic &lt;trysoI&gt;</td>
<td>-3000010</td>
</tr>
</tbody>
</table>

---

### 5.36.4.166 heuristics/twoopt

---
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/twoopt/freq</td>
<td>frequency for calling primal heuristic <code>&lt;twoopt&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/twoopt/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;twoopt&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/twoopt/intopt</td>
<td>Should Integer-2-Optimization be applied or not?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/twoopt/matchingrate</td>
<td>parameter to determine the percentage of rows two variables have to share before they are considered equal</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/twoopt/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;twoopt&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/twoopt/maxnslaves</td>
<td>maximum number of slaves for one master variable</td>
<td>199</td>
</tr>
<tr>
<td>heuristics/twoopt/priority</td>
<td>priority of heuristic <code>&lt;twoopt&gt;</code></td>
<td>-20100</td>
</tr>
<tr>
<td>heuristics/twoopt/waitingnodes</td>
<td>user parameter to determine number of nodes to wait after last best solution before calling heuristic</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.167 heuristics/undercover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/undercover/fixingalts</td>
<td>prioritized sequence of fixing values used ('l'lp relaxation, 'u'lp relaxation, 'i'ncumbent solution)</td>
<td>li</td>
</tr>
<tr>
<td>heuristics/undercover/freq</td>
<td>frequency for calling primal heuristic <code>&lt;undercover&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;undercover&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/undercover/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/undercover/onlyconvexify</td>
<td>should we only fix variables in order to obtain a convex problem?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/postnlp</td>
<td>should the NLP heuristic be called to polish a feasible solution?</td>
<td>1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/undercover/beforecuts</td>
<td>should the heuristic be called at root node before cut separation?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/conflictweight</td>
<td>weight for conflict score in fixing order</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/coverbd</td>
<td>should bounddisjunction constraints be covered (or just copied)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/coveringobj</td>
<td>objective function of the covering problem</td>
<td>u</td>
</tr>
<tr>
<td></td>
<td>(influenced nonlinear 'c'ontaints/'t'erms, 'd'omain size, 'l'ocks, 'u'nit of up/down locks, 'm'in of up/down locks)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: c, d, l, m, t, u</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/cutoffweight</td>
<td>weight for cutoff score in fixing order</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/fixingorder</td>
<td>order in which variables should be fixed (increasing 'C'onflict score, decreasing 'c'onflict score, increasing 'V'ariable index, decreasing 'v'ariable index)</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>Range: C, c, V, v</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/fixintfirst</td>
<td>should integer variables in the cover be fixed first?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/inferenceweight</td>
<td>weight for inference score in fixing order</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/locksrounding</td>
<td>shall LP values for integer vars be rounded according to locks?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxbacktracks</td>
<td>maximum number of backtracks in fix-and-propagate</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxcoversizeconss</td>
<td>maximum coversize (as ratio to the percentage of non-affected constraints)</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxcoversizevars</td>
<td>maximum coversize (as fraction of total number of variables)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;undercover&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxrecovers</td>
<td>maximum number of recoverings</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/undercover/maxreorders</td>
<td>maximum number of reorderings of the fixing order</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

#### 5.36.4.168 heuristics/vbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/vbounds/freq</td>
<td>frequency for calling primal heuristic &lt;vbounds&gt; (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;vbounds&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/minintfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/minmipfixingrate</td>
<td>minimum percentage of variables that have to be fixed within sub-SCIP (integer and continuous)</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/feasvariant</td>
<td>which variants of the vbounds heuristic that try to stay feasible should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound)</td>
<td>6</td>
</tr>
</tbody>
</table>
### heuristics/vbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/vbounds/maxbacktracks</td>
<td>maximum number of backtracks during the fixing process</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;vbounds&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/maxnodes</td>
<td>maximum number of nodes to regard in the sub-problem</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/maxproprounds</td>
<td>maximum number of propagation rounds during probing (-1: infinity)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/minimprove</td>
<td>factor by which vbounds heuristic should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: {0, 1}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/priority</td>
<td>priority of heuristic &lt;vbounds&gt;</td>
<td>2500</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/tightenvariant</td>
<td>which tightening variants of the vbounds heuristic should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 7}</td>
<td></td>
</tr>
<tr>
<td>heuristics/vbounds/uselockfixings</td>
<td>should more variables be fixed based on variable locks if the fixing rate was not reached?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### heuristics/veclendiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/veclendiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>heuristics/veclendiving/freq</td>
<td>frequency for calling primal heuristic &lt;veclendiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/veclendiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;veclendiving&gt;</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>heuristics/veclendiving/lpresolvedomchgquot</td>
<td>percentage of immediate domain changes during probing to trigger LP resolve</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/veclendiving/lpsolvefreq</td>
<td>LP solve frequency for diving heuristics (0: only after enough domain changes have been found)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>heuristics/veclendiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/veclending/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/veclending/onlylpbranchcands</td>
<td>should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/veclending/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;veclending&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/veclending/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclending/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclending/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/veclending/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/veclending/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/veclending/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclending/priority</td>
<td>priority of heuristic &lt;veclending&gt;</td>
<td>-1003100</td>
</tr>
</tbody>
</table>

### 5.36.4.170 heuristics/zeroobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/zeroobj/freq</td>
<td>frequency for calling primal heuristic &lt;zeroobj&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/zeroobj/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;zeroobj&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/zeroobj/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/zeroobj/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heuristics/zeroobj/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| `heuristics/zeroobj/maxdepth` | maximal depth level to call primal heuristic \(<\text{zeroobj}>\) \((-1: \text{no limit})\)  
Range: \([-1, \ldots, 65534]\)                                                                 | 0       |
| `heuristics/zeroobj/maxlpiters` | maximum number of LP iterations to be performed in the subproblem  
Range: \([-1, \ldots, \infty]\)                                                                 | 5000    |
| `heuristics/zeroobj/maxnodes` | maximum number of nodes to regard in the subproblem  
Range: \([0, \ldots, \infty]\)                                                                 | 1000    |
| `heuristics/zeroobj/minimprove` | factor by which zeroobj should at least improve the incumbent  
Range: \([0, 1]\)                                                                 | 0.01    |
| `heuristics/zeroobj/minnodes` | minimum number of nodes required to start the subproblem  
Range: \([0, \ldots, \infty]\)                                                                 | 100     |
| `heuristics/zeroobj/onlywithoutsol` | should heuristic only be executed if no primal solution was found, yet?  
Range: boolean                                                                 | 1       |
| `heuristics/zeroobj/priority` | priority of heuristic \(<\text{zeroobj}>\)  
Range: \([-536870912, \ldots, 536870911]\)                                                                 | 100     |
| `heuristics/zeroobj/useuct` | should uct node selection be used at the beginning of the search?  
Range: boolean                                                                 | 0       |

### 5.36.4.171 `heuristics/zirounding`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `heuristics/zirounding/freq`  | frequency for calling primal heuristic \(<\text{zirounding}>\) \((-1: \text{never}, 0: \text{only at depth freqofs})\)  
Range: \([-1, \ldots, 65534]\)                                                                 | 1       |
| `heuristics/zirounding/freqofs` | frequency offset for calling primal heuristic \(<\text{zirounding}>\)  
Range: \([0, \ldots, 65534]\)                                                                 | 0       |

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| `heuristics/zirounding/maxdepth` | maximal depth level to call primal heuristic \(<\text{zirounding}>\) \((-1: \text{no limit})\)  
Range: \([-1, \ldots, 65534]\)                                                                 | -1      |
| `heuristics/zirounding/maxroundingloops` | determines maximum number of rounding loops  
Range: \([-1, \ldots, \infty]\)                                                                 | 2       |
| `heuristics/zirounding/minstopncalls` | determines the minimum number of calls before percentage-based deactivation of Zirounding is applied  
Range: \([1, \ldots, \infty]\)                                                                 | 1000    |
| `heuristics/zirounding/priority` | priority of heuristic \(<\text{zirounding}>\)  
Range: \([-536870912, \ldots, 536870911]\)                                                                 | -500    |
| `heuristics/zirounding/stoppercentage` | if percentage of found solutions falls below this parameter, Zirounding will be deactivated  
Range: \([0, 1]\)                                                                 | 0.02    |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/zirounding/stopziround</td>
<td>flag to determine if Zirounding is deactivated after a certain percentage of unsuccessful calls</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.172 history

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>history/allowmerge</td>
<td>should variable histories be merged from sub-SCIPs whenever possible?</td>
<td>0</td>
</tr>
<tr>
<td>history/allowtransfer</td>
<td>should variable histories be transferred to initialize SCIP copies?</td>
<td>0</td>
</tr>
<tr>
<td>history/valuebased</td>
<td>should statistics be collected for variable domain value pairs?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.173 limits

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>limits/absgap</td>
<td>solving stops, if the absolute gap (</td>
<td>\text{primalbound} - \text{dualbound}</td>
</tr>
<tr>
<td>limits/autorestartnodes</td>
<td>if solve exceeds this number of nodes for the first time, an automatic restart is triggered (-1: no automatic restart)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/bestsol</td>
<td>solving stops, if the given number of solution improvements were found (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/gap</td>
<td>solving stops, if the relative gap ( \frac{</td>
<td>\text{primal} - \text{dual}</td>
</tr>
<tr>
<td>limits/maxorigsol</td>
<td>maximal number of solutions candidates to store in the solution storage of the original problem</td>
<td>10</td>
</tr>
<tr>
<td>limits/maxsol</td>
<td>maximal number of solutions to store in the solution storage</td>
<td>100</td>
</tr>
<tr>
<td>limits/memory</td>
<td>maximal memory usage in MB; reported memory usage is lower than real memory usage!</td>
<td>GAMS workspace</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>limits/nodes</td>
<td>maximal number of nodes to process (-1: no limit)</td>
<td>GAMS nodlim, if &gt; 0, otherwise -1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>limits/restarts</td>
<td>solving stops, if the given number of restarts was triggered (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>limits/softtime</td>
<td>soft time limit which should be applied after first solution was found</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>(-1.0: disabled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ∞]</td>
<td></td>
</tr>
<tr>
<td>limits/solutions</td>
<td>solving stops, if the given number of solutions were found (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>limits/stallnodes</td>
<td>solving stops, if the given number of nodes was processed since the last</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>improvement of the primal solution value (-1: no limit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>limits/time</td>
<td>maximal time in seconds to run</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>limits/totalnodes</td>
<td>maximal number of total nodes (incl. restarts) to process (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.3.6.4.174  lp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp/alwaysgetduals</td>
<td>should the Farkas duals always be collected when an LP is found to be</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>infeasible?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/initialalgorithm</td>
<td>LP algorithm for solving initial LP relaxations (automatic 's' simplex,</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>'p' primal simplex, 'd' dual simplex, 'b' barrier with 'c' crossover)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: s, p, d, b, c</td>
<td></td>
</tr>
<tr>
<td>lp/pricing</td>
<td>LP pricing strategy ('l'pi default, 'a'uto, 'f'ull pricing, 'p'artial,</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>'s' steepest edge pricing, 'q' quickstart steepest edge pricing, 'd' even</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pricing)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: l, a, f, p, s, q, d</td>
<td></td>
</tr>
<tr>
<td>lp/resolvealgorithm</td>
<td>LP algorithm for resolving LP relaxations if a starting basis exists</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>(automatic 's' simplex, 'p' primal simplex, 'd' dual simplex,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'b' barrier with 'c' crossover)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: s, p, d, b, c</td>
<td></td>
</tr>
<tr>
<td>lp/solvedepth</td>
<td>maximal depth for solving LP at the nodes (-1: no depth limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>lp/solvefreq</td>
<td>frequency for solving LP at the nodes (-1: never; 0: only root LP) Range: {-1, ..., 65534}</td>
<td>1</td>
</tr>
<tr>
<td>lp/solver</td>
<td>LP solver to use (clp, cplex, soplex) Range: string</td>
<td>cplex, if licensed, otherwise soplex</td>
</tr>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lp/checkdualfeas</td>
<td>should LP solutions be checked for dual feasibility, resolving LP when numerical troubles occur? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/checkfarkas</td>
<td>should infeasibility proofs from the LP be checked? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/checkprimfeas</td>
<td>should LP solutions be checked for primal feasibility, resolving LP when numerical troubles occur? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/checkstability</td>
<td>should LP solver's return status be checked for stability? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/cleanupcols</td>
<td>should new non-basic columns be removed after LP solving? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>lp/cleanupcolsroot</td>
<td>should new non-basic columns be removed after root LP solving? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>lp/cleanuprows</td>
<td>should new basic rows be removed after LP solving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/cleanuprowsroot</td>
<td>should new basic rows be removed after root LP solving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/clearinitialprobinglp</td>
<td>should lp state be cleared at the end of probing mode when lp was initially unsolved, e.g., when called right after presolving? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>lp/colagelimit</td>
<td>maximum age a dynamic column can reach before it is deleted from the LP (-1: don't delete columns due to aging) Range: {-1, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>lp/conditionlimit</td>
<td>maximum condition number of LP basis counted as stable (-1.0: no limit) Range: [-1, ∞]</td>
<td>-1</td>
</tr>
<tr>
<td>lp/disablecutoff</td>
<td>disable the cutoff bound in the LP solver? (0: enabled, 1: disabled, 2: auto) Range: {0, ..., 2}</td>
<td>2</td>
</tr>
<tr>
<td>lp/fastmip</td>
<td>which FASTMIP setting of LP solver should be used? 0: off, 1: low Range: {0, ..., 1}</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>lp/freesolvalbuffers</td>
<td>should the buffers for storing LP solution values during diving be freed at end of diving?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/iterlim</td>
<td>iteration limit for each single LP solve (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ..., ∞]</td>
<td></td>
</tr>
<tr>
<td>lp/lexdualalgo</td>
<td>should the lexicographic dual algorithm be used?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/lexdualbasic</td>
<td>choose fractional basic variables in lexicographic dual algorithm?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/lexdualmaxrounds</td>
<td>maximum number of rounds in the lexicographic dual algorithm (-1: unbounded)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ..., ∞]</td>
<td></td>
</tr>
<tr>
<td>lp/lexdualrootonly</td>
<td>should the lexicographic dual algorithm be applied only at the root node</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/lexdualstalling</td>
<td>turn on the lex dual algorithm only when stalling?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/minmarkowitz</td>
<td>minimal Markowitz threshold to control sparsity/stability in LU factorization</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0.0001, 0.9999]</td>
<td></td>
</tr>
<tr>
<td>lp/presolving</td>
<td>should presolving of LP solver be used?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/refactorinterval</td>
<td>LP refactorization interval (0: auto)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>lp/resolvetimerfac</td>
<td>factor of average LP iterations that is used as LP iteration limit for LP resolve (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ∞]</td>
<td></td>
</tr>
<tr>
<td>lp/resolvetimermin</td>
<td>minimum number of iterations that are allowed for LP resolve</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>lp/resolverestore</td>
<td>should the LP be resolved to restore the state at start of diving (if FALSE we buffer the solution values)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>lp/rootiterlim</td>
<td>iteration limit for initial root LP solve (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ..., ∞]</td>
<td></td>
</tr>
<tr>
<td>lp/rowagelimit</td>
<td>maximum age a dynamic row can reach before it is deleted from the LP (-1: don’t delete rows due to aging)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36 SCIP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lp/rowrepswitch</strong></td>
<td>simplex algorithm shall use row representation of the basis if number of rows divided by number of columns exceeds this value (-1.0 to disable row representation)</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Range: $[-1, \infty)$</td>
<td></td>
</tr>
<tr>
<td><strong>lp/scaling</strong></td>
<td>LP scaling (0: none, 1: normal, 2: aggressive)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., 2}$</td>
<td></td>
</tr>
<tr>
<td><strong>lp/solutionpolishing</strong></td>
<td>LP solution polishing method (0: disabled, 1: only root, 2: always, 3: auto)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., 3}$</td>
<td></td>
</tr>
<tr>
<td><strong>lp/threads</strong></td>
<td>number of threads used for solving the LP (0: automatic)</td>
<td>GAMS threads</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., 64}$</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.175 memory

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>memory/savefac</strong></td>
<td>fraction of maximal memory usage resulting in switch to memory saving mode</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, 1]$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/arraygrowfac</strong></td>
<td>memory growing factor for dynamically allocated arrays</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Range: $[1, 10]$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/arraygrowinit</strong></td>
<td>initial size of dynamically allocated arrays</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., \infty}$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/pathgrowfac</strong></td>
<td>memory growing factor for path array</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: $[1, 10]$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/pathgrowinit</strong></td>
<td>initial size of path array</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., \infty}$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/treegrowfac</strong></td>
<td>memory growing factor for tree array</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: $[1, 10]$</td>
<td></td>
</tr>
<tr>
<td><strong>memory/treegrowinit</strong></td>
<td>initial size of tree array</td>
<td>65536</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, ..., \infty}$</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.176 misc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>misc/allowstrongdualreds</strong></td>
<td>should strong dual reductions be allowed in propagation and presolving?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td><strong>misc/allowweakdualreds</strong></td>
<td>should weak dual reductions be allowed in propagation and presolving?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td><strong>misc/avoidmemout</strong></td>
<td>try to avoid running into memory limit by restricting plugins like heuristics?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>misc/calcintegral</td>
<td>should SCIP calculate the primal dual integral value?</td>
<td>1</td>
</tr>
<tr>
<td>misc/catchctrlc</td>
<td>should the CTRL-C interrupt be caught by SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>misc/estimexternmem</td>
<td>should the usage of external memory be estimated?</td>
<td>1</td>
</tr>
<tr>
<td>misc/finitesolutionstore</td>
<td>should SCIP try to remove infinite fixings from solutions copied to the solution store?</td>
<td>0</td>
</tr>
<tr>
<td>misc/improvingssols</td>
<td>should only solutions be checked which improve the primal bound</td>
<td>0</td>
</tr>
<tr>
<td>misc/outputorigsol</td>
<td>should the best solution be transformed to the original space and be output in command line run?</td>
<td>1</td>
</tr>
<tr>
<td>misc/printreason</td>
<td>should the reason be printed if a given start solution is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>misc/referencevalue</td>
<td>objective value for reference purposes</td>
<td>∞</td>
</tr>
<tr>
<td>misc/resetstat</td>
<td>should the statistics be reset if the transformed problem is freed (in case of a Benders' decomposition this parameter should be set to FALSE)</td>
<td>1</td>
</tr>
<tr>
<td>misc/scaleobj</td>
<td>should the objective function be scaled so that it is always integer?</td>
<td>1</td>
</tr>
<tr>
<td>misc/transorigsols</td>
<td>should SCIP try to transfer original solutions to the transformed space (after presolving)?</td>
<td>1</td>
</tr>
<tr>
<td>misc/transsolsorig</td>
<td>should SCIP try to transfer transformed solutions to the original space (after solving)?</td>
<td>1</td>
</tr>
<tr>
<td>misc/useconstable</td>
<td>should a hashtable be used to map from constraint names to constraints?</td>
<td>1</td>
</tr>
<tr>
<td>misc/usesmalltables</td>
<td>should smaller hashtables be used? yields better performance for small problems with about 100 variables</td>
<td>0</td>
</tr>
<tr>
<td>misc/usesymmetry</td>
<td>bitset describing used symmetry handling technique (0: off; 1: polyhedral (orbitopes and/or symresacks); 2: orbital fixing; 3: orbitopes and orbital fixing; 4: Schreier Sims cuts; 5: Schreier Sims cuts and orbitopes); 6: Schreier Sims cuts and orbital fixing; 7: Schreier Sims cuts, orbitopes, and orbital fixing</td>
<td>7</td>
</tr>
<tr>
<td>misc/usevartable</td>
<td>should a hashtable be used to map from variable names to variables?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nlhdlr/bilinear/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/bilinear/maxsepadepth</td>
<td>maximum depth to apply separation</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/bilinear/maxseparounds</td>
<td>maximum number of separation rounds in a local node</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/bilinear/maxseparoundsroot</td>
<td>maximum number of separation rounds in the root node</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/bilinear/useinteval</td>
<td>whether to use the interval evaluation callback of the nlhdlr</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/bilinear/usereverseprop</td>
<td>whether to use the reverse propagation callback of the nlhdlr</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

**5.36.4.178 nlhdlr/concave**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/concave/detectsum</td>
<td>whether to run convexity detection when the root of an expression is a sum</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/concave/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nlhdlr/concave/cvxprodcomp</td>
<td>whether to use convexity check on product composition (f(h) + h)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/concave/cvxquadratic</td>
<td>whether to use convexity check on quadratics</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/concave/cvxsignomial</td>
<td>whether to use convexity check on signomials</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/concave/handletrivial</td>
<td>whether to also handle trivial convex expressions</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

**5.36.4.179 nlhdlr/convex**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/convex/detectsum</td>
<td>whether to run convexity detection when the root of an expression is a non-quadratic sum</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/convex/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/convex/extendedform</td>
<td>whether to create extended formulations instead of looking for maximal convex expressions</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/convex/cvxprodcomp</td>
<td>whether to use convexity check on product composition ( f(h) + h )</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/convex/cvxquadratic</td>
<td>whether to use convexity check on quadratics</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/convex/cvxsignomial</td>
<td>whether to use convexity check on signomials</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/convex/handletrivial</td>
<td>whether to also handle trivial convex expressions</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.180 nlhdlr/default

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/default/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.181 nlhdlr/perspective

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/perspective/adjrefpoint</td>
<td>whether to adjust the reference point</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/perspective/convexonly</td>
<td>whether perspective cuts are added only for convex expressions</td>
<td>0</td>
</tr>
<tr>
<td>nlhdlr/perspective/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/perspective/maxproprounds</td>
<td>maximal number of propagation rounds in probing</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/perspective/mindomreduction</td>
<td>minimal relative reduction in a variable's domain for applying probing</td>
<td>0.1</td>
</tr>
<tr>
<td>nlhdlr/perspective/minviolprobing</td>
<td>minimal violation w.r.t. auxiliary variables for applying probing</td>
<td>1e-05</td>
</tr>
<tr>
<td>nlhdlr/perspective/probingfreq</td>
<td>probing frequency (-1 - no probing, 0 - root node only)</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/perspective/probingonlyinsepa</td>
<td>whether to do probing only in separation</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/perspective/tightenbounds</td>
<td>whether variable semicontinuity is used to tighten variable bounds</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.182 nlhdlr/quadratic

---
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/quadratic/atwhichnodes</td>
<td>determines at which nodes cut is used (if it’s -1, it’s used only at the root node, if it’s n ≥ 0, it’s used at every multiple of n)</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/quadratic/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/quadratic/ignorebadrayrestriction</td>
<td>should cut be generated even with bad numerics when restricting to ray?</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/quadratic/ignorenhighre</td>
<td>should cut be added even when range / efficacy is large?</td>
<td>1</td>
</tr>
<tr>
<td>nlhdlr/quadratic/maxrank</td>
<td>maximal rank a slackvar can have</td>
<td>∞</td>
</tr>
<tr>
<td>nlhdlr/quadratic/mincutviolation</td>
<td>minimal cut violation the generated cuts must fulfill to be added to the LP</td>
<td>0.0001</td>
</tr>
<tr>
<td>nlhdlr/quadratic/minviolation</td>
<td>minimal violation the constraint must fulfill such that a cut is generated</td>
<td>0.0001</td>
</tr>
<tr>
<td>nlhdlr/quadratic/ncutslimit</td>
<td>limit for number of cuts generated consecutively</td>
<td>2</td>
</tr>
<tr>
<td>nlhdlr/quadratic/ncutslimiteroot</td>
<td>limit for number of cuts generated at root node</td>
<td>20</td>
</tr>
<tr>
<td>nlhdlr/quadratic/nstrengthlimit</td>
<td>limit for number of rays we do the strengthening for</td>
<td>∞</td>
</tr>
<tr>
<td>nlhdlr/quadratic/useboundsasrays</td>
<td>use bounds of variables in quadratic as rays for intersection cuts</td>
<td>0</td>
</tr>
<tr>
<td>nlhdlr/quadratic/useintersectioncuts</td>
<td>whether to use intersection cuts for quadratic constraints to separate</td>
<td>0</td>
</tr>
<tr>
<td>nlhdlr/quadratic/usestrengthening</td>
<td>whether the strengthening should be used</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.183 nlhdlr/quotient

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/quotient/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.184 nlhdlr/soc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlhdlr/soc/compeigenvalues</td>
<td>Should Eigenvalue computations be done to detect complex cases in quadratic constraints?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nlhdlr/soc/enabled</td>
<td>should this nonlinear handler be used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nlhdlr/soc/mincutefficacy</td>
<td>Minimum efficacy which a cut needs in order to be added.</td>
<td>1e-05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.185 nlp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp/disable</td>
<td>should the NLP relaxation be always disabled (also for NLPs/MINLPs)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.186 nlpi/ipopt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlpi/ipopt/hessian_approximation</td>
<td>Indicates what Hessian information is to be used. Valid values if not empty: exact limited-memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/hsllib</td>
<td>Name of library containing HSL routines for load at runtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/linear_solver</td>
<td>Linear solver used for step computations. Valid values if not empty: ma27 ma57 ma77 ma86 ma97 pardiso pardisomkl mumps custom</td>
<td>ma27, if IpoptH licensed, mumps otherwise</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/linear_system_scaling</td>
<td>Method for scaling the linear system. Valid values if not empty: none mc19 slack-based</td>
<td>mc19, if IpoptH licensed, empty otherwise</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/mu_strategy</td>
<td>Update strategy for barrier parameter. Valid values if not empty: monotone adaptive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/nlp_scaling_method</td>
<td>Select the technique used for scaling the NLP. Valid values if not empty: none user-scaling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gradient-based equilibration-based</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/optfile</td>
<td>name of Ipopt options file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/pardisolib</td>
<td>Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>nlpi/ipopt/print_level</td>
<td>Output verbosity level. -1 to use NLPI or Ipopt default</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 12}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.187 nodeselection

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/childsel</td>
<td>child selection rule ('d'own, 'u'p, 'p'sudo costs, 'i'nference, 'l'p value, 'r'oot LP value difference, 'h'ybrid inference/root LP value difference)</td>
<td>h</td>
</tr>
</tbody>
</table>

### 5.36.4.188 nodeselection/bfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/bfs/stdpriority</td>
<td>priority of node selection rule <code>&lt;bfs&gt;</code> in standard mode</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {−536870912, ..., 1073741823}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodeselection/bfs/maxplungedepth</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {−1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/bfs/maxplungequot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/bfs/memsavepriority</td>
<td>priority of node selection rule <code>&lt;bfs&gt;</code> in memory saving mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {−536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/bfs/minplungedepth</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {−1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.189 nodeselection/breadthfirst

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/breadthfirst/stdpriority</td>
<td>priority of node selection rule <code>&lt;breadthfirst&gt;</code> in standard mode</td>
<td>-10000</td>
</tr>
<tr>
<td></td>
<td>Range: {−536870912, ..., 1073741823}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodeselection/breadthfirst/memsavepriority</td>
<td>priority of node selection rule <code>&lt;breadthfirst&gt;</code> in memory saving mode</td>
<td>-100000</td>
</tr>
<tr>
<td></td>
<td>Range: {−536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>
5.36.4.190  node selection/dfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/dfs/stdpriority</td>
<td>priority of node selection rule &lt;dfs&gt; in standard mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 1073741823}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/dfs/memsavepriority</td>
<td>priority of node selection rule &lt;dfs&gt; in memory saving mode</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.191  node selection/estimate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/estimate/bestnodefreq</td>
<td>frequency at which the best node instead of the best estimate is selected (0: never)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/breadthfirstdepth</td>
<td>depth until breadth-first search is applied</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/plungeoffset</td>
<td>number of nodes before doing plunging the first time</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/stdpriority</td>
<td>priority of node selection rule &lt;estimate&gt; in standard mode</td>
<td>200000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 1073741823}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/estimate/maxplungedepth</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/maxplungequot</td>
<td>maximal quotient (estimate - lower-bound)/(cutoffbound - lowerbound) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/memsavepriority</td>
<td>priority of node selection rule &lt;estimate&gt; in memory saving mode</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/minplungedepth</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.192  node selection/hybridestim

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/hybridestim/bestnodefreq</td>
<td>frequency at which the best node instead of the hybrid best estimate / best bound is selected (0: never)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>nodeselection/hybridestim/stdpriority</td>
<td>priority of node selection rule in standard mode</td>
<td>50000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 1073741823}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/hybridestim/estimweight</td>
<td>weight of estimate value in node selection score (0: pure best bound search, 1: pure best estimate search)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>nodeselection/hybridestim/maxplungedepth</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ([-1, \infty])</td>
<td></td>
</tr>
<tr>
<td>nodeselection/hybridestim/maxplungequot</td>
<td>maximal quotient ((\text{estimate} - \text{lower-bound})/(\text{cutoffbound} - \text{lowerbound})) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>nodeselection/hybridestim/memsavepriority</td>
<td>priority of node selection rule in memory saving mode</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>nodeselection/hybridestim/minplungedepth</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ([-1, \infty])</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.193 nodeselection/restartdfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/restartdfs/countonleyeaves</td>
<td>count only leaf nodes (otherwise all nodes)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>nodeselection/restartdfs/selectbestfreq</td>
<td>frequency for selecting the best node instead of the deepest one</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: ({0, \ldots, \infty})</td>
<td></td>
</tr>
<tr>
<td>nodeselection/restartdfs/stdpriority</td>
<td>priority of node selection rule in standard mode</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 1073741823}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/restartdfs/memsavepriority</td>
<td>priority of node selection rule in memory saving mode</td>
<td>50000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.194 nodeselection/uct

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/uct/stdpriority</td>
<td>priority of node selection rule in standard mode</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 1073741823}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/uct/memsavepriority</td>
<td>priority of node selection rule in memory saving mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.195 numerics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>numerics/checkfeastolfac</td>
<td>feasibility tolerance factor; for checking the feasibility of the best solution Range: [0, ( \infty )]</td>
<td>1</td>
</tr>
<tr>
<td>numerics/dualfeastol</td>
<td>feasibility tolerance for reduced costs in LP solution Range: [1e-17, 0.001]</td>
<td>1e-07</td>
</tr>
<tr>
<td>numerics/epsilon</td>
<td>absolute values smaller than this are considered zero Range: [1e-20, 0.001]</td>
<td>1e-09</td>
</tr>
<tr>
<td>numerics/feastol</td>
<td>feasibility tolerance for constraints Range: [1e-17, 0.001]</td>
<td>1e-06</td>
</tr>
<tr>
<td>numerics/lpfeastolfactor</td>
<td>factor w.r.t. primal feasibility tolerance that determines default (and maximal) primal feasibility tolerance of LP solver Range: [1e-06, 1]</td>
<td>1</td>
</tr>
<tr>
<td>numerics/sumepsilon</td>
<td>absolute values of sums smaller than this are considered zero Range: [1e-17, 0.001]</td>
<td>1e-06</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>numerics/barrierconvtol</td>
<td>LP convergence tolerance used in barrier algorithm Range: [1e-17, 0.001]</td>
<td>1e-10</td>
</tr>
<tr>
<td>numerics/boundstreps</td>
<td>minimal relative improve for strengthening bounds Range: [1e-17, ( \infty )]</td>
<td>0.05</td>
</tr>
<tr>
<td>numerics/hugeval</td>
<td>values larger than this are considered huge and should be handled separately (e.g., in activity computation) Range: [0, ( \infty )]</td>
<td>1e+15</td>
</tr>
<tr>
<td>numerics/pseudocostdelta</td>
<td>minimal objective distance value to use for branching pseudo cost updates Range: [0, ( \infty )]</td>
<td>0.0001</td>
</tr>
<tr>
<td>numerics/pseudocosteps</td>
<td>minimal variable distance value to use for branching pseudo cost updates Range: [1e-17, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>numerics/recomputefac</td>
<td>minimal decrease factor that causes the recomputation of a value (e.g., pseudo objective) instead of an update Range: [0, ( \infty )]</td>
<td>1e+07</td>
</tr>
</tbody>
</table>

### 5.36.4.196 presolving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/maxrestarts</td>
<td>maximal number of restarts (-1: unlimited) Range: {-1, \ldots, \infty}</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>presolving/maxrounds</td>
<td>maximal number of presolving rounds (-1: unlimited, 0: off)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td>presolving/abortfac</td>
<td>abort presolve, if at most this fraction of the problem was changed in last presolve round</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/clqtablefac</td>
<td>limit on number of entries in clique table relative to number of problem nonzeros</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/donotaggr</td>
<td>should aggregation of variables be forbidden?</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/donotmultaggr</td>
<td>should multi-aggregation of variables be forbidden?</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/immrestartfac</td>
<td>fraction of integer variables that were fixed in the root node triggering an immediate restart with preprocessing</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/restartfac</td>
<td>fraction of integer variables that were fixed in the root node triggering a restart with preprocessing after root node evaluation</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/restartminred</td>
<td>minimal fraction of integer variables removed after restart to allow for an additional restart</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/subrestartfac</td>
<td>fraction of integer variables that were globally fixed during the solving process triggering a restart with preprocessing</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>5.36.4.197 presolving/boundshift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>presolving/boundshift/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td>presolving/boundshift/flipping</td>
<td>is flipping allowed (multiplying with -1)?</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/boundshift/integer</td>
<td>shift only integer ranges?</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/boundshift/maxshift</td>
<td>absolute value of maximum shift</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/boundshift/priority</td>
<td>priority of presolver &lt;boundshift&gt;</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>presolving/boundshift/timing</td>
<td>timing mask of presolver &lt;boundshift&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>5.36.4.198 presolving/convertinttobin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>presolving/convertinttobin/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/convertinttobin/maxdomainsize</td>
<td>absolute value of maximum domain size for converting an integer variable to binaries variables</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>presolving/convertinttobin/onlypoweroftwo</td>
<td>should only integer variables with a domain size of $2^p - 1$ be converted, there we don't need an knapsack-constraint for restricting the sum of the binaries</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>presolving/convertinttobin/priority</td>
<td>priority of presolver &lt;convertinttobin&gt;</td>
<td>6000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>presolving/convertinttobin/samelocksinbotherdirections</td>
<td>should only integer variables with uplocks equals downlocks be converted</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>presolving/convertinttobin/timing</td>
<td>timing mask of presolver &lt;convertinttobin&gt;</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>(4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.199 presolving/domcol

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/domcol/continuousred</td>
<td>should reductions for continuous variables be performed?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>presolving/domcol/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>presolving/domcol/mummaxpairs</td>
<td>maximal number of pair comparisons</td>
<td>1048576</td>
</tr>
<tr>
<td></td>
<td>Range: {1024, ..., 1000000000}</td>
<td></td>
</tr>
<tr>
<td>presolving/domcol/muminminpairs</td>
<td>minimal number of pair comparisons</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>Range: {100, ..., 1048576}</td>
<td></td>
</tr>
<tr>
<td>presolving/domcol/predbndstr</td>
<td>should predictive bound strengthening be applied?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/domcol/priority</td>
<td>priority of presolver &lt;domcol&gt;</td>
<td>-1000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>presolving/domcol/timing</td>
<td>timing mask of presolver &lt;domcol&gt;</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>(4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.200 presolving/dualagg
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualagg/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>presolving/dualagg/priority</td>
<td>priority of presolver (&lt;\text{dualagg}&gt;) Range: {-536870912, ..., 536870911}</td>
<td>-12000</td>
</tr>
<tr>
<td>presolving/dualagg/timing</td>
<td>timing mask of presolver (&lt;\text{dualagg}&gt;) (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>16</td>
</tr>
</tbody>
</table>

5.36.4.201 presolving/dualcomp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualcomp/componlydisvars</td>
<td>should only discrete variables be compensated? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>presolving/dualcomp/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>-1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>presolving/dualcomp/priority</td>
<td>priority of presolver (&lt;\text{dualcomp}&gt;) Range: {-536870912, ..., 536870911}</td>
<td>-50</td>
</tr>
<tr>
<td>presolving/dualcomp/timing</td>
<td>timing mask of presolver (&lt;\text{dualcomp}&gt;) (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>16</td>
</tr>
</tbody>
</table>

5.36.4.202 presolving/dualinfer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualinfer/maxdualbndloops</td>
<td>maximal number of dual bound strengthening loops Range: {-1, ..., \infty}</td>
<td>12</td>
</tr>
<tr>
<td>presolving/dualinfer/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>0</td>
</tr>
<tr>
<td>presolving/dualinfer/maxrowsupport</td>
<td>Maximum number of row's non-zeros for changing inequality to equality Range: {2, ..., \infty}</td>
<td>3</td>
</tr>
<tr>
<td>presolving/dualinfer/twocolcombine</td>
<td>use convex combination of columns for determining dual bounds Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>presolving/dualinfer/maxcombinefails</td>
<td>maximal number of consecutive useless column combines Range: {-1, ..., \infty}</td>
<td>1000</td>
</tr>
<tr>
<td>presolving/dualinfer/maxconsiderednonzeros</td>
<td>maximal number of considered non-zeros within one column (-1: no limit) Range: {-1, ..., \infty}</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>presolving/dualinfer/maxhashfac</td>
<td>Maximum number of hashlist entries as multiple of number of columns in the problem (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualinfer/maxpairfac</td>
<td>Maximum number of processed column pairs as multiple of the number of columns in the problem (-1: no limit)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualinfer/maxretrievefails</td>
<td>maximal number of consecutive useless hashtable retrieves</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualinfer/priority</td>
<td>priority of presolver &lt;dualinfer&gt;</td>
<td>-3000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualinfer/timing</td>
<td>timing mask of presolver &lt;dualinfer&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Range: {4, ..., 60}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.203 presolving/dualsparsify

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualsparsify/maxbinfillin</td>
<td>maximal fillin for binary variables (-1: unlimited)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/maxcontfillin</td>
<td>maximal fillin for continuous variables (-1: unlimited)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/maxintfillin</td>
<td>maximal fillin for integer variables including binaries (-1: unlimited)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/mineliminatednonzeros</td>
<td>minimal eliminated nonzeros within one column if we need to add a constraint to the problem</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., $\infty$}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualsparsify/enablecopy</td>
<td>should dualsparsify presolver be copied to sub-SCIPs?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/maxconsiderednonzeros</td>
<td>maximal number of considered nonzeros within one column (-1: no limit)</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., $\infty$}</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/maxretrievefac</td>
<td>limit on the number of useless vs. useful hashtable retrieves as a multiple of the number of constraints</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: [0, $\infty$]</td>
<td></td>
</tr>
<tr>
<td>presolving/dualsparsify/preservegoodlocks</td>
<td>should we preserve good locked properties of variables (at most one lock in one direction)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.204 presolving/gateextraction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/gateextraction/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: (-1, ..., \infty)</td>
<td>-1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>presolving/gateextraction/onlysetpart</td>
<td>should we only try to extract set-partitioning constraints and no and-constraints Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>presolving/gateextraction/priority</td>
<td>priority of presolver (&lt;\text{gateextraction}\rangle) Range: {-536870912, ..., 536870911}</td>
<td>1000000</td>
</tr>
<tr>
<td>presolving/gateextraction/searchequations</td>
<td>should we try to extract set-partitioning constraint out of one logico and one corresponding set-packing constraint Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/gateextraction/sorting</td>
<td>order logico constraints to extract big-gates before smaller ones (-1), do not order them (0) or order them to extract smaller gates at first (1) Range: {-1, ..., 1}</td>
<td>1</td>
</tr>
<tr>
<td>presolving/gateextraction/timing</td>
<td>timing mask of presolver (&lt;\text{gateextraction}\rangle) ((4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)) Range: {4, ..., 60}</td>
<td>16</td>
</tr>
</tbody>
</table>

### 5.36.4.205 presolving/implies

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/implies/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>-1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>presolving/implies/priority</td>
<td>priority of presolver (&lt;\text{implies}\rangle) Range: {-536870912, ..., 536870911}</td>
<td>-10000</td>
</tr>
</tbody>
</table>
### 5.36.4.206 presolving/inttobinary

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/inttobinary/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/inttobinary/priority</td>
<td>priority of presolver &lt;inttobinary&gt; Range: {-536870912, ..., 536870911}</td>
<td>7000000</td>
</tr>
<tr>
<td>presolving/inttobinary/timing</td>
<td>timing mask of presolver &lt;inttobinary&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>4</td>
</tr>
</tbody>
</table>

### 5.36.4.207 presolving/milp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/milp/hugebound</td>
<td>absolute bound value that is considered too huge for activity based calculations Range: [0, ∞]</td>
<td>1e+08</td>
</tr>
<tr>
<td>presolving/milp/markowitztolerance</td>
<td>the markowitz tolerance used for substitutions Range: [0, 1]</td>
<td>0.01</td>
</tr>
<tr>
<td>presolving/milp/maxfillinpersubstitution</td>
<td>maximal possible fillin for substitutions to be considered Range: {-2147483648, ..., ∞}</td>
<td>3</td>
</tr>
<tr>
<td>presolving/milp/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/milp/modifyconsfac</td>
<td>modify SCIP constraints when the number of nonzeros or rows is at most this factor times the number of nonzeros or rows before presolving Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>presolving/milp/randomseed</td>
<td>the random seed used for randomization of tie breaking Range: {-2147483648, ..., ∞}</td>
<td>0</td>
</tr>
<tr>
<td>presolving/milp/threads</td>
<td>maximum number of threads presolving may use (0: automatic) Range: {0, ..., ∞}</td>
<td>GAMS threads</td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/milp/enabledomcol</td>
<td>should the dominated column presolver be enabled within the presolve library? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/milp/enabledualinfer</td>
<td>should the dualinfer presolver be enabled within the presolve library? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>presolving/milp/enablemultiaggr</td>
<td>should the multi-aggregation presolver be enabled within the presolve library? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/milp/enableparallelrows</td>
<td>should the parallel rows presolver be enabled within the presolve library? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/milp/enableprobing</td>
<td>should the probing presolver be enabled within the presolve library? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/milp/enablesparsify</td>
<td>should the sparsify presolver be enabled within the presolve library? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>presolving/milp/maxshiftperrow</td>
<td>maximal amount of nonzeros allowed to be shifted to make space for substitutions Range: {0, ..., \infty}</td>
<td>10</td>
</tr>
<tr>
<td>presolving/milp/priority</td>
<td>priority of presolver (&lt;milp&gt;) Range: {-536870912, ..., 536870911}</td>
<td>9999999</td>
</tr>
<tr>
<td>presolving/milp/probfilename</td>
<td>filename to store the problem before MILP presolving starts Range: string</td>
<td>-</td>
</tr>
<tr>
<td>presolving/milp/timing</td>
<td>timing mask of presolver (&lt;milp&gt;) (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>8</td>
</tr>
</tbody>
</table>

### 5.36.4.208 presolving/qpkktref

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/qpkktref/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>0</td>
</tr>
<tr>
<td>presolving/qpkktref/addkktbinary</td>
<td>if TRUE then allow binary variables for KKT update Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>presolving/qpkktref/priority</td>
<td>priority of presolver (&lt;qpkktref&gt;) Range: {-536870912, ..., 536870911}</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/qpkktref/timing</td>
<td>timing mask of presolver (&lt;qpkktref&gt;) (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>8</td>
</tr>
<tr>
<td>presolving/qpkktref/updatequadbounded</td>
<td>if TRUE then only apply the update to QPs with bounded variables; if the variables are not bounded then a finite optimal solution might not exist and the KKT conditions would then be invalid Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/qpkktref/updatequadindef</td>
<td>if TRUE then apply quadratic constraint update even if the quadratic constraint matrix is known to be indefinite Range: boolean</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.36.4.209 presolving/redvub

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/redvub/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

| presolving/redvub/priority                  | priority of presolver <redvub>                                              | -9000000 |
|                                             | Range: {-536870912, ..., 536870911}                                         |         |
| presolving/redvub/timing                    | timing mask of presolver <redvub> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) | 16      |
|                                             | Range: {4, ..., 60}                                                         |         |

### 5.36.4.210 presolving/sparsify

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/sparsify/maxbinfillin</td>
<td>maximal fillin for binary variables (-1: unlimited)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>presolving/sparsify/maxcontfillin</td>
<td>maximal fillin for continuous variables (-1: unlimited)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>presolving/sparsify/maxintfillin</td>
<td>maximal fillin for integer variables including binaries (-1: unlimited)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>presolving/sparsify/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<p>| presolving/sparsify/cancellinear           | should we cancel nonzeros in constraints of the linear constraint handler? | 1       |
|                                           | Range: boolean                                                             |         |
| presolving/sparsify/enablecopy            | should sparsify presolver be copied to sub-SCIps?                          | 1       |
|                                           | Range: boolean                                                             |         |
| presolving/sparsify/maxconsiderednonzeros | maximal number of considered non-zeros within one row (-1: no limit)       | 70      |
|                                           | Range: {-1, ..., ∞}                                                         |         |
| presolving/sparsify/maxnonzeros           | maximal support of one equality to be used for cancelling (-1: no limit)   | -1      |
|                                           | Range: {-1, ..., ∞}                                                         |         |
| presolving/sparsify/maxretrievecfac       | limit on the number of useless vs. useful hashtable retrieves as a multiple of the number of constraints | 100     |
|                                           | Range: [0, ∞]                                                              |         |
| presolving/sparsify/preserveintcoeffs     | should we forbid cancellations that destroy integer coefficients?           | 1       |
|                                           | Range: boolean                                                             |         |
| presolving/sparsify/priority              | priority of presolver &lt;sparsify&gt;                                           | -24000  |
|                                           | Range: {-536870912, ..., 536870911}                                         |         |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/sparsify/rowsort</td>
<td>order in which to process inequalities ('n' = sorting, 'i' = increasing nonzeros, 'd' = decreasing nonzeros)</td>
<td>d</td>
</tr>
<tr>
<td>presolving/sparsify/timing</td>
<td>timing mask of presolver &lt;sparsify&gt; (4:FAST, 8: MEDIUM, 16: EXHAUSTIVE, 32: FINAL)</td>
<td>16</td>
</tr>
<tr>
<td>presolving/sparsify/waitingfac</td>
<td>number of calls to wait until next execution as a multiple of the number of useless calls</td>
<td>2</td>
</tr>
</tbody>
</table>

### 5.36.4.211 presolving/stuffing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/stuffing/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>presolving/stuffing/priority</td>
<td>priority of presolver &lt;stuffing&gt;</td>
<td>-100</td>
</tr>
<tr>
<td>presolving/stuffing/timing</td>
<td>timing mask of presolver &lt;stuffing&gt; (4:FAST, 8: MEDIUM, 16: EXHAUSTIVE, 32: FINAL)</td>
<td>16</td>
</tr>
</tbody>
</table>

### 5.36.4.212 presolving/trivial

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/trivial/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/trivial/priority</td>
<td>priority of presolver &lt;trivial&gt;</td>
<td>9000000</td>
</tr>
<tr>
<td>presolving/trivial/timing</td>
<td>timing mask of presolver &lt;trivial&gt; (4:FAST, 8: MEDIUM, 16: EXHAUSTIVE, 32: FINAL)</td>
<td>4</td>
</tr>
</tbody>
</table>

### 5.36.4.213 presolving/tworowbnd

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/tworowbnd/maxcombinefails</td>
<td>maximal number of consecutive useless row combines</td>
<td>1000</td>
</tr>
<tr>
<td>presolving/tworowbnd/maxconsiderednonzeros</td>
<td>maximal number of considered non-zeros within one row (-1: no limit)</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>presolving/tworowbnd/maxhashfac</td>
<td>Maximum number of hashlist entries as multiple of number of rows in the problem (-1: no limit) Range: {-1, ..., \infty}</td>
<td>10</td>
</tr>
<tr>
<td>presolving/tworowbnd/maxpairfac</td>
<td>Maximum number of processed row pairs as multiple of the number of rows in the problem (-1: no limit) Range: {-1, ..., \infty}</td>
<td>1</td>
</tr>
<tr>
<td>presolving/tworowbnd/maxretrievefails</td>
<td>maximal number of consecutive useless hashtable retrieves Range: {-1, ..., \infty}</td>
<td>1000</td>
</tr>
<tr>
<td>presolving/tworowbnd/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/tworowbnd/enablecopy</td>
<td>should tworowbnd presolver be copied to sub-SCIPs? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>presolving/tworowbnd/priority</td>
<td>priority of presolver &lt;tworowbnd&gt; Range: {-536870912, ..., 536870911}</td>
<td>-2000</td>
</tr>
<tr>
<td>presolving/tworowbnd/timing</td>
<td>timing mask of presolver &lt;tworowbnd&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}</td>
<td>16</td>
</tr>
</tbody>
</table>

5.36.4.214 propagating

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/abortoncutoff</td>
<td>should propagation be aborted immediately? setting this to FALSE could help conflict analysis to produce more conflict constraints Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>propagating/maxrounds</td>
<td>maximal number of propagation rounds per node (-1: unlimited) Range: {-1, ..., \infty}</td>
<td>100</td>
</tr>
<tr>
<td>propagating/maxroundsroot</td>
<td>maximal number of propagation rounds in the root node (-1: unlimited) Range: {-1, ..., \infty}</td>
<td>1000</td>
</tr>
</tbody>
</table>

5.36.4.215 propagating/dualfix

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/dualfix/freq</td>
<td>frequency for calling propagator &lt;dualfix&gt; (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>0</td>
</tr>
<tr>
<td>propagating/dualfix/maxprerounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., \infty}</td>
<td>-1</td>
</tr>
</tbody>
</table>

Options for expert users
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/dualfix/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/dualfix/presolpriority</td>
<td>presolving priority of propagator &lt;dualfix&gt;</td>
<td>8000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/dualfix/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;dualfix&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>propagating/dualfix/priority</td>
<td>priority of propagator &lt;dualfix&gt;</td>
<td>8000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/dualfix/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS))</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.216 propagating/genvbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/genvbounds/freq</td>
<td>frequency for calling propagator &lt;genvbounds&gt; (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/genvbounds/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/global</td>
<td>apply global propagation?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/presolpriority</td>
<td>presolving priority of propagator &lt;genvbounds&gt;</td>
<td>-2000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;genvbounds&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/priority</td>
<td>priority of propagator &lt;genvbounds&gt;</td>
<td>3000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/propasconss</td>
<td>should genvbounds be transformed to (linear) constraints?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/propinrootnode</td>
<td>apply genvbounds in root node if no new incumbent was found?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/genvbounds/sort</td>
<td>sort genvbounds and wait for bound change events?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/genvbounds/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>15</td>
</tr>
</tbody>
</table>

5.36.4.217 propagating/nlobbt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/nlobbt/addlrows</td>
<td>should non-initial LP rows be used?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/nlobbt/freq</td>
<td>frequency for calling propagator &lt;nlobbt&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/nlobbt/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/nlobbt/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/nlobbt/feastolfac</td>
<td>factor for NLP feasibility tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>propagating/nlobbt/itlimitfactor</td>
<td>LP iteration limit for nlobbt will be this factor times total LP iterations in root node</td>
<td>2</td>
</tr>
<tr>
<td>propagating/nlobbt/minlinearfrac</td>
<td>minimum (convex nlrows)/(linear nlrows) threshold to apply propagator</td>
<td>0.02</td>
</tr>
<tr>
<td>propagating/nlobbt/minnonconvexfrac</td>
<td>(convex nlrows)/(nonconvex nlrows) threshold to apply propagator</td>
<td>0.2</td>
</tr>
<tr>
<td>propagating/nlobbt/nlpiterlimit</td>
<td>iteration limit of NLP solver; 0 for no limit</td>
<td>500</td>
</tr>
<tr>
<td>propagating/nlobbt/nlptimelimit</td>
<td>time limit of NLP solver; 0.0 for no limit</td>
<td>0</td>
</tr>
<tr>
<td>propagating/nlobbt/nlpverblevel</td>
<td>verbosity level of NLP solver</td>
<td>0</td>
</tr>
<tr>
<td>propagating/nlobbt/presolpriority</td>
<td>presolving priority of propagator &lt;nlobbt&gt;</td>
<td>0</td>
</tr>
<tr>
<td>propagating/nlobbt/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;nlobbt&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td>propagating/nlobbt/priority</td>
<td>priority of propagator &lt;nlobbt&gt;</td>
<td>-1100000</td>
</tr>
<tr>
<td>propagating/nlobbt/relobjtolfac</td>
<td>factor for NLP relative objective tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>propagating/nlobbt/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>4</td>
</tr>
</tbody>
</table>
### 5.36.4.218 propagating/obbt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/obbt/boundstreps</td>
<td>minimal relative improve for strengthening bounds</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/conditionlimit</td>
<td>maximum condition limit used in LP solver (-1.0: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: [-1, ∞]</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/dualfeasitol</td>
<td>feasibility tolerance for reduced costs used in obbt; this value is used if SCIP’s dual feasitol is greater</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/freq</td>
<td>frequency for calling propagator &lt;obbt&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/itlimitfactor</td>
<td>multiple of root node LP iterations used as total LP iteration limit for obbt (≤ 0: no limit)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/itlimitfactorbilin</td>
<td>multiple of OBBT LP limit used as total LP iteration limit for solving bilinear inequality LPs (&lt; 0 for no limit)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/minitlimit</td>
<td>minimum LP iteration limit</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/minnonconvexity</td>
<td>minimum absolute value of nonconvex eigenvalues for a bilinear term</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

**Options for expert users**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/obbt/applyfilterrounds</td>
<td>try to filter bounds in so-called filter rounds by solving auxiliary LPs?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/applytrivialfilter</td>
<td>try to filter bounds with the LP solution after each solve?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/creatabilinineqs</td>
<td>solve auxiliary LPs in order to find valid inequalities for bilinear terms?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/creategenvbounds</td>
<td>should obbt try to provide genvbounds if possible?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/createlinccons</td>
<td>create linear constraints from inequalities for bilinear terms?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/genvbdsduringfilter</td>
<td>should we try to generate genvbounds during trivial and aggressive filtering?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/obbt/genvbdsduringsepa</td>
<td>try to create genvbounds during separation process?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/minfilter</td>
<td>minimal number of filtered bounds to apply another filter round</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/normalize</td>
<td>should coefficients in filtering be normalized w.r.t. the domains sizes?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/onlynonconvexvars</td>
<td>only apply obbt on non-convex variables</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/orderingalgo</td>
<td>select the type of ordering algorithm which should be used (0: no special ordering, 1: greedy, 2: greedy reverse)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/presolpriority</td>
<td>presolving priority of propagator &lt;obbt&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;obbt&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/priority</td>
<td>priority of propagator &lt;obbt&gt;</td>
<td>-1000000</td>
</tr>
<tr>
<td></td>
<td>Range: {536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/propagatefreq</td>
<td>trigger a propagation round after that many bound tightenings (0: no propagation)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/sepanaxiter</td>
<td>maximum number of iteration spend to separate an obbt LP solution</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/sepanaxiter</td>
<td>minimum number of iteration spend to separate an obbt LP solution</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/separatesol</td>
<td>should the obbt LP solution be separated?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/tightcontboundsprobing</td>
<td>should continuous bounds be tightened during the probing mode?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/tightintboundsprobing</td>
<td>should integral bounds be tightened during the probing mode?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/obbt/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS))</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.219 propagating/probing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/probing/freq</td>
<td>frequency for calling propagator &lt;probing&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>
### Option Description Default

#### propagating/probing/maxprerounds
- maximal number of presolving rounds the propagator participates in
- (-1: no limit)
- Range: \([-1, \ldots, \infty]\)
- \(-1\)

#### propagating/probing/maxruns
- maximal number of runs, probing participates in
- (-1: no limit)
- Range: \([-1, \ldots, \infty]\)
- \(1\)

### Options for expert users

#### propagating/probing/delay
- should propagator be delayed, if other propagators found reductions?
- Range: boolean
- \(1\)

#### propagating/probing/maxdepth
- maximal depth until propagation is executed
- (-1: no limit)
- Range: \([-1, \ldots, \infty]\)
- \(-1\)

#### propagating/probing/maxfixings
- maximal number of fixings found, until probing is interrupted
- (0: don't interrupt)
- Range: \([0, \ldots, \infty]\)
- \(25\)

#### propagating/probing/maxnumuseless
- maximal number of probings without fixings, until probing is aborted
- (0: don't abort)
- Range: \([0, \ldots, \infty]\)
- \(0\)

#### propagating/probing/maxsumuseless
- maximal number of successive probings without fixings, bound changes, and implications, until probing is aborted
- Range: \([0, \ldots, \infty]\)
- \(50\)

#### propagating/probing/maxtotaluseless
- maximal number of successive probings without fixings, until probing is aborted
- Range: \([0, \ldots, \infty]\)
- \(1000\)

#### propagating/probing/presolpriority
- presolving priority of propagator
- \(<\text{probing}>\)
- Range: \([-536870912, \ldots, 536870911]\)
- \(-100000\)

#### propagating/probing/presoltiming
- timing mask of the presolving method of propagator
- \(<\text{probing}>\)
- (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)
- Range: \([2, \ldots, 60]\)
- \(16\)

#### propagating/probing/priority
- priority of propagator
- \(<\text{probing}>\)
- Range: \([-536870912, \ldots, 536870911]\)
- \(-100000\)

#### propagating/probing/proprounds
- maximal number of propagation rounds in probing subproblems
- (-1: no limit, 0: auto)
- Range: \([-1, \ldots, \infty]\)
- \(-1\)

#### propagating/probing/timingmask
- timing when propagator should be called
- (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)
- Range: \([1, \ldots, 15]\)
- \(4\)

---

### 5.36.4.220 propagating/pseudoobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| propagating/pseudoobj/freq | frequency for calling propagator \(<\text{pseudoobj}>\)
- (-1: never, 0: only in root node)
- Range: \([-1, \ldots, 65534]\) | \(1\) |
| propagating/pseudoobj/maxprerounds | maximal number of presolving rounds the propagator participates in
- (-1: no limit)
- Range: \([-1, \ldots, \infty]\) | \(-1\) |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/pseudoobj/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/force</td>
<td>should the propagator be forced even if active pricer are present?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/maximplvars</td>
<td>maximum number of binary variables the implications are used if turned on (-1: unlimited)?</td>
<td>50000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/maxnewvars</td>
<td>number of variables added after the propagator is reinitialized?</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/maxvarsfrac</td>
<td>maximal fraction of non-binary variables with non-zero objective without a bound reduction before aborted</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/minuseless</td>
<td>minimal number of successive non-binary variable propagations without a bound reduction before aborted</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/presolpriority</td>
<td>presolving priority of propagator &lt;pseudoobj&gt;</td>
<td>6000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;pseudoobj&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/priority</td>
<td>priority of propagator &lt;pseudoobj&gt;</td>
<td>3000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/propcutoffbound</td>
<td>propagate new cutoff bound directly globally</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/propfullinroot</td>
<td>whether to propagate all non-binary variables when we are propagating the root node</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/propuseimplics</td>
<td>use implications to strengthen the propagation of binary variable (increasing the objective change)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/respropuseimplics</td>
<td>use implications to strengthen the resolve propagation of binary variable (increasing the objective change)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/pseudoobj/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS))</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 15}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.221 propagating/redcost
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/redcost/continuous</td>
<td>should reduced cost fixing be also applied to continuous variables?</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>propagating/redcost/freq</td>
<td>frequency for calling propagator &lt;redcost&gt; (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/redcost/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/redcost/useimplics</td>
<td>should implications be used to strength the reduced cost for binary variables?</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>propagating/redcost/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/force</td>
<td>should the propagator be forced even if active pricer are present?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/presolpriority</td>
<td>presolving priority of propagator &lt;redcost&gt;</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/presoltiming</td>
<td>timing mask of the presolving method of propagator &lt;redcost&gt; (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>28</td>
</tr>
<tr>
<td>propagating/redcost/priority</td>
<td>priority of propagator &lt;redcost&gt;</td>
<td>1000000</td>
</tr>
<tr>
<td>propagating/redcost/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS))</td>
<td>6</td>
</tr>
</tbody>
</table>

5.36.4.222 propagating/rootredcost

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/rootredcost/freq</td>
<td>frequency for calling propagator &lt;rootredcost&gt; (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/rootredcost/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>propagating/rootredcost/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/rootredcost/force</td>
<td>should the propagator be forced even if active pricer are present?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/rootredcost/onlybinary</td>
<td>should only binary variables be propagated?</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| propagating/rootredcost/presolpriority | presolving priority of propagator  \(<\text{rootredcost}>\)  
Range: \{-536870912, ..., 536870911\} | 0       |
| propagating/rootredcost/presoltiming  | timing mask of the presolving method of propagator  \(<\text{rootredcost}>\)  
\(4: \text{FAST}, 8: \text{MEDIUM}, 16: \text{EXHAUSTIVE}, 32: \text{FINAL}\)  
Range: \{2, ..., 60\} | 28      |
| propagating/rootredcost/priority      | priority of propagator  \(<\text{rootredcost}>\)  
Range: \{-536870912, ..., 536870911\} | 10000000 |
| propagating/rootredcost/timingmask    | timing when propagator should be called  
\(1: \text{BEFORELP}, 2: \text{DURINGLPLOOP}, 4: \text{AFTERLPLOOP}, 15: \text{ALWAYS}\)  
Range: \{1, ..., 15\} | 5       |

5.36.4.223 propagating/symmetry

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| propagating/symmetry/freq             | frequency for calling propagator  \(<\text{symmetry}>\)  
\(-1: \text{never}, 0: \text{only in root node}\)  
Range: \{-1, ..., 65534\} | 1       |
| propagating/symmetry/maxprerounds     | maximal number of presolving rounds the propagator participates in  \(-1: \text{no limit}\)  
Range: \{-1, ..., \infty\} | -1      |

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| propagating/symmetry/addconflictcuts  | Should Schreier Sims constraints be added if we use a conflict based rule?  
Range: boolean | 1       |
| propagating/symmetry/addconsstiming   | timing of adding constraints  \(0 = \text{before presolving}, 1 = \text{during presolving}, 2 = \text{after presolving}\)  
Range: \{0, ..., 2\} | 2       |
| propagating/symmetry/addstrongsbcs    | Should strong SBCs for enclosing orbit of symmetric subgroups be added if orbitopes are not used?  
Range: boolean | 0       |
| propagating/symmetry/addsymresacks    | Add inequalities for symresacks for each generator?  
Range: boolean | 0       |
| propagating/symmetry/addweaksbcs      | Should we add weak SBCs for enclosing orbit of symmetric subgroups?  
Range: boolean | 1       |
| propagating/symmetry/checksymmetries  | Should all symmetries be checked after computation?  
Range: boolean | 0       |
| propagating/symmetry/compresssymmetries | Should non-affected variables be removed from permutation to save memory?  
Range: boolean | 1       |
| propagating/symmetry/compressothreshold | Compression is used if percentage of moved vars is at most the threshold.  
Range: \[0, 1\] | 0.5     |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/symmetry/conssaddlp</td>
<td>Should the symmetry breaking constraints be added to the LP?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/detectorbitopes</td>
<td>Should we check whether the components of the symmetry group can be handled by orbitopes?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/detectsubgroups</td>
<td>Should we try to detect symmetric subgroups of the symmetry group on binary variables?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/displaynorbitvars</td>
<td>Should the number of variables affected by some symmetry be displayed?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/doubleequations</td>
<td>Double equations to positive/negative version?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/maxgenerators</td>
<td>limit on the number of generators that should be produced within symmetry detection (0 = no limit)</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., }</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/maxnconsssubgroup</td>
<td>maximum number of constraints up to which subgroup structures are detected</td>
<td>500000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., }</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/ofsymcomptiming</td>
<td>timing of symmetry computation for orbital fixing (0 = before presolving, 1 = during presolving, 2 = at first call)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/onlybinarysymmetry</td>
<td>Is only symmetry on binary variables used?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/performpresolving</td>
<td>run orbital fixing during presolving?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/preferlessrows</td>
<td>Shall orbitopes with less rows be preferred in detection?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/presolpriority</td>
<td>presolving priority of propagator (&lt;\text{symmetry}&gt;)</td>
<td>-1000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/presoltiming</td>
<td>timing mask of the presolving method of propagator (&lt;\text{symmetry}&gt;) (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 60}</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/priority</td>
<td>priority of propagator (&lt;\text{symmetry}&gt;)</td>
<td>-1000000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/recomputerstart</td>
<td>recomputes symmetries after a restart has occurred? (0 = never, 1 = always, 2 = if OF found reduction)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>propagating/symmetry/sstaddcuts</td>
<td>Should Schreier Sims constraints be added?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/symmetry/sstleaderrule</td>
<td>rule to select the leader in an orbit (0: first var; 1: last var; 2: var having most conflicting vars in orbit; 3: var having most conflicting vars in problem)</td>
<td>0</td>
</tr>
<tr>
<td>propagating/symmetry/sstleadervartype</td>
<td>bitset encoding which variable types can be leaders (1: binary; 2: integer; 4: impl. int; 8: continuous); if multiple types are allowed, take the one with most affected vars</td>
<td>14</td>
</tr>
<tr>
<td>propagating/symmetry/sstmixedcomponents</td>
<td>Should Schreier Sims constraints be added if a symmetry component contains variables of different types?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/symmetry/ssttiebreakrule</td>
<td>rule to select the orbit in Schreier Sims inequalities (variable in 0: minimum size orbit; 1: maximum size orbit; 2: orbit with most variables in conflict with leader)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/symmetry/symfixnonbinaryvars</td>
<td>Whether all non-binary variables shall be not affected by symmetries if OF is active?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/symmetry/timingmask</td>
<td>timing when propagator should be called (1: BEFORELP, 2: DURINGLPLOOP, 4: AFTERLPLOOP, 15: ALWAYS))</td>
<td>1</td>
</tr>
<tr>
<td>propagating/symmetry/usecolumnsparsity</td>
<td>Should the number of cons a variable is contained in be exploited in symmetry detection?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/symmetry/usedynamicprop</td>
<td>whether dynamic propagation should be used for full orbitopes</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.224 propagating/vbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/vbounds/detectcycles</td>
<td>should cycles in the variable bound graph be identified?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/dotoposort</td>
<td>should the bounds be topologically sorted in advance?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/vbounds/freq</td>
<td>frequency for calling propagator &lt;vbounds&gt; (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/vbounds/maxcliquesexhaustive</td>
<td>maximum number of cliques per variable to run clique table analysis in exhaustive presolving</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/vbounds/maxcliquesmedium</td>
<td>maximum number of cliques per variable to run clique table analysis in medium presolving Range: ([0, \infty])</td>
<td>50</td>
</tr>
<tr>
<td>propagating/vbounds/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in ((-1: \text{no limit})) Range: ([-1, \infty])</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/vbounds/minnewcliques</td>
<td>minimum percentage of new cliques to trigger another clique table analysis Range: ([0, 1])</td>
<td>0.1</td>
</tr>
<tr>
<td>propagating/vbounds/usebdwidening</td>
<td>should bound widening be used to initialize conflict analysis? Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/vbounds/delay</td>
<td>should propagator be delayed, if other propagators found reductions? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/presolpriority</td>
<td>presolving priority of propagator (&lt;\text{vbounds}&gt;) Range: ([-536870912, ..., 536870911])</td>
<td>-90000</td>
</tr>
<tr>
<td>propagating/vbounds/presoltiming</td>
<td>timing mask of the presolving method of propagator (&lt;\text{vbounds}&gt;) ((4: \text{FAST}, 8: \text{MEDIUM}, 16: \text{EXHAUSTIVE}, 32: \text{FINAL})) Range: ([2, ..., 60])</td>
<td>24</td>
</tr>
<tr>
<td>propagating/vbounds/priority</td>
<td>priority of propagator (&lt;\text{vbounds}&gt;) Range: ([-536870912, ..., 536870911])</td>
<td>3000000</td>
</tr>
<tr>
<td>propagating/vbounds/sortcliques</td>
<td>should cliques be regarded for the topological sort? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/timingmask</td>
<td>timing when propagator should be called ((1: \text{BEFORELP}, 2: \text{DURINGLPLOOP}, 4: \text{AFTERLPLOOP}, 15: \text{ALWAYS})) Range: ([1, ..., 15])</td>
<td>5</td>
</tr>
<tr>
<td>propagating/vbounds/usecliques</td>
<td>should cliques be propagated? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/useimplics</td>
<td>should implications be propagated? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/usevbounds</td>
<td>should vbounds be propagated? Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.225 randomization

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomization/lpseed</td>
<td>random seed for LP solver, e.g. for perturbations in the simplex ((0: \text{LP default})) Range: ([0, ..., \infty])</td>
<td>0</td>
</tr>
<tr>
<td>randomization/permutationseed</td>
<td>seed value for permuting the problem after reading/transformation ((0: \text{no permutation})) Range: ([0, ..., \infty])</td>
<td>0</td>
</tr>
<tr>
<td>randomization/randomseedshift</td>
<td>global shift of all random seeds in the plugins and the LP random seed Range: ([0, ..., \infty])</td>
<td>0</td>
</tr>
</tbody>
</table>
### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomization/permutecons</td>
<td>should order of constraints be permuted (depends on permutationseed)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>randomization/permutevars</td>
<td>should order of variables be permuted (depends on permutationseed)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.226 separating

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separation (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/maxcoefratio</td>
<td>maximal ratio between coefficients in strongcg, cmir, and flowcover cuts</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: [1, (\infty)]</td>
<td></td>
</tr>
<tr>
<td>separating/maxcoefratiofacrowprep</td>
<td>maximal ratio between coefficients (as factor of 1/feastol) to ensure in rowprep cleanup</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [0, (\infty)]</td>
<td></td>
</tr>
<tr>
<td>separating/maxcuts</td>
<td>maximal number of cuts separated per separation round (0: disable local separation)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/maxcutsroot</td>
<td>maximal number of separated cuts at the root node (0: disable root node separation)</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/maxlocalbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying local separation (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/maxstallrounds</td>
<td>maximal number of consecutive separation rounds without objective or integrality improvement in local nodes (-1: no additional restriction)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/maxstallroundsroot</td>
<td>maximal number of consecutive separation rounds without objective or integrality improvement in the root node (-1: no additional restriction)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., (\infty}}</td>
<td></td>
</tr>
<tr>
<td>separating/minactivityquot</td>
<td>minimum cut activity quotient to convert cuts into constraints during a restart (0.0: all cuts are converted)</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/minfficacy</td>
<td>minimal efficacy for a cut to enter the LP</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, \infty]$</td>
<td></td>
</tr>
<tr>
<td>separating/minfficacyroot</td>
<td>minimal efficacy for a cut to enter the LP in the root node</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>Range: $[0, \infty]$</td>
<td></td>
</tr>
<tr>
<td>separating/poolfreq</td>
<td>separation frequency for the global cut pool (-1: disable global cut pool, 0: only separate pool at the root)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: $[-1, \ldots, 65534]$</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/cutagelimit</td>
<td>maximum age a cut can reach before it is deleted from the global cut pool, or -1 to keep all cuts</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>separating/cutselrestart</td>
<td>cut selection during restart ('a'ge, activity 'q'uotient)</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>Range: a, q</td>
<td></td>
</tr>
<tr>
<td>separating/cutselsubscip</td>
<td>cut selection for sub SCIPs ('a'ge, activity 'q'uotient)</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>Range: a, q</td>
<td></td>
</tr>
<tr>
<td>separating/efficacynorm</td>
<td>row norm to use for efficacy calculation ('e'ucilcane, 'm'aximum, 's'um, 'd'iscrete)</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>Range: e, m, s, d</td>
<td></td>
</tr>
<tr>
<td>separating/filtercutpoolrel</td>
<td>should cutpool separate only cuts with high relative efficacy?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/maxaddrounds</td>
<td>maximal additional number of separation rounds in subsequent price-and-cut loops (-1: no additional restriction)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>separating/maxroundsrootsubrun</td>
<td>maximal number of separation rounds in the root node of a subsequent run (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>separating/maxruns</td>
<td>maximal number of runs for which separation is enabled (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>separating/orthofunc</td>
<td>function used for calc. scalar prod. in orthogonality test ('e'ucilcane, 'd'iscrete)</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>Range: e, d</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.227 separating/aggregation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/aggregation/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/freq</td>
<td>frequency for calling separator &lt;aggregation&gt; (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, 65534}$</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxrounds</td>
<td>maximal number of cmir separation rounds per node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: ${-1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/aggregation/maxroundsroot</td>
<td>maximal number of cmir separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxsepacuts</td>
<td>maximal number of cmir cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxsepacutroot</td>
<td>maximal number of cmir cuts separated per separation round in the root node</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/aggrtol</td>
<td>tolerance for bound distances used to select continuous variable in current</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>aggregated constraint to be eliminated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/densityoffset</td>
<td>additional number of variables allowed in row on top of density</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/densityscore</td>
<td>weight of row density in the aggregation scoring of the rows</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;aggregation&gt;</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>is called (1: call at each multiple of frequency)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/fixintegralrhs</td>
<td>should an additional variable be complemented if f0 = 0?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxaggdensity</td>
<td>maximal density of aggregated row</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxaggrs</td>
<td>maximal number of aggregations for each row per separation round</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxaggrsroot</td>
<td>maximal number of aggregations for each row per separation round in the root</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>node</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>compared to best node's dual bound for applying separator &lt;aggregation&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0.0: only on current best node, 1.0: on all nodes)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxfails</td>
<td>maximal number of consecutive unsuccessful aggregation tries (-1: unlimited)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxfailsroot</td>
<td>maximal number of consecutive unsuccessful aggregation tries in the root</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>node (-1: unlimited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>
### 5.36.4.228 separating/aggregation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/aggregation/maxrowdensity</td>
<td>maximal density of row to be used in aggregation</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxrowfac</td>
<td>maximal row aggregation factor</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxslack</td>
<td>maximal slack of rows to be used in aggregation</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxslackroot</td>
<td>maximal slack of rows to be used in aggregation in the root node</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxtestdelta</td>
<td>maximal number of different deltas to try (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxtries</td>
<td>maximal number of rows to start aggregation with per separation round</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>(-1: unlimited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/maxtriesroot</td>
<td>maximal number of rows to start aggregation with per separation round in</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>the root node (-1: unlimited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/priority</td>
<td>priority of separator &lt;aggregation&gt;</td>
<td>-3000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/slackscore</td>
<td>weight of slack in the aggregation scoring of the rows</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/aggregation/trynegscaling</td>
<td>should negative values also be tested in scaling?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.228 separating/cgmip

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/cgmip/addviolationcons</td>
<td>add constraint to subscip that only allows violated cuts (otherwise add obj. limit)?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/addviolconshdlr</td>
<td>add constraint handler to filter out violated cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/cmirownbounds</td>
<td>tell CMIR-generator which bounds to used in rounding?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/conshdlrusesenorm</td>
<td>should the violation constraint handler use the norm of a cut to check for feasibility?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/contconvert</td>
<td>Convert some integral variables to be continuous to reduce the size of the sub-MIP?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/contconvfrac</td>
<td>fraction of integral variables converted to be continuous (if contconvert)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/cgmip/contconvmin</td>
<td>minimum number of integral variables before some are converted to be continuous</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/decisiontree</td>
<td>Use decision tree to turn separation on/off?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/earlyterm</td>
<td>terminate separation if a violated (but possibly sub-optimal) cut has been found?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/freq</td>
<td>frequency for calling separator <code>&lt;cgmip&gt;</code> (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/genprimalsols</td>
<td>Try to generate primal solutions from Gomory cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/intconvert</td>
<td>Convert some integral variables attaining fractional values to have integral value?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/intconvfrac</td>
<td>fraction of frac. integral variables converted to have integral value (if intconvert)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: {0, 1}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/intconvmin</td>
<td>minimum number of integral variables before some are converted to have integral value</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/maxdepth</td>
<td>maximal depth at which the separator is applied (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/maxnodelimit</td>
<td>maximum number of nodes considered for sub-MIP (-1: unlimited)</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/maxrounds</td>
<td>maximal number of cgmip separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/maxroundsroot</td>
<td>maximal number of cgmip separation rounds in the root node (-1: unlimited)</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/maxrowage</td>
<td>maximal age of rows to consider if onlyactive rows is false</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/minnodelimit</td>
<td>minimum number of nodes considered for sub-MIP (-1: unlimited)</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/objlone</td>
<td>Should the objective of the sub-MIP minimize the l1-norm of the multipliers?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/objweightsize</td>
<td>Weight each row by its size?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/onlyactive</td>
<td>Use only active rows to generate cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/onlyintvars</td>
<td>Generate cuts for problems with only integer variables?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/cgmip/onlyrankone</td>
<td>Separate only rank 1 inequalities w.r.t. CG-MIP separator?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/output</td>
<td>Should information about the sub-MIP and cuts be displayed?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/primalseparation</td>
<td>only separate cuts that are tight for the best feasible solution?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/skipmultbounds</td>
<td>Skip the upper bounds on the multipliers in the sub-MIP?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/subscipfast</td>
<td>Should the settings for the sub-MIP be optimized for speed?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/usecmir</td>
<td>use CMIR-generator (otherwise add cut directly)?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/usecutpool</td>
<td>use cutpool to store CG-cuts even if they are not efficient?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/useobjlb</td>
<td>Use lower bound on objective function (via primal solution)?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/useobjub</td>
<td>Use upper bound on objective function (via primal solution)?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/usestrongcg</td>
<td>use strong CG-function to strengthen cut?</td>
<td>0</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/cgmip/cutcoefbnd</td>
<td>bounds on the values of the coefficients in the CG-cut</td>
<td>1000</td>
</tr>
<tr>
<td>separating/cgmip/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;cgmip&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/cgmip/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;cgmip&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/memorylimit</td>
<td>memory limit for sub-MIP</td>
<td>∞</td>
</tr>
<tr>
<td>separating/cgmip/objweight</td>
<td>weight used for the row combination coefficient in the sub-MIP objective</td>
<td>0.001</td>
</tr>
<tr>
<td>separating/cgmip/priority</td>
<td>priority of separator &lt;cgmip&gt;</td>
<td>-1000</td>
</tr>
<tr>
<td>separating/cgmip/timelimit</td>
<td>time limit for sub-MIP</td>
<td>∞</td>
</tr>
</tbody>
</table>
5.36.4.229  separating/clique

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/clique/freq</td>
<td>frequency for calling separator &lt;clique&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/maxsepacuts</td>
<td>maximal number of clique cuts separated per separation round (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/clique/backtrackfreq</td>
<td>frequency for premature backtracking up to tree level 1 (0: no backtracking)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/cliquedensity</td>
<td>minimal density of cliques to use a dense clique table</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/clique/cliquetablemem</td>
<td>maximal memory size of dense clique table (in kb)</td>
<td>20000</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2.09715e+06]</td>
<td></td>
</tr>
<tr>
<td>separating/clique/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/clique/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;clique&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;clique&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/clique/maxtreenodes</td>
<td>maximal number of nodes in branch and bound tree (-1: no limit)</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/maxzeroextensions</td>
<td>maximal number of zero-valued variables extending the clique (-1: no limit)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/priority</td>
<td>priority of separator &lt;clique&gt;</td>
<td>-5000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>separating/clique/scaleval</td>
<td>factor for scaling weights</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.230  separating/closecuts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/closecuts/freq</td>
<td>frequency for calling separator &lt;closecuts&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/closecuts/closethres</td>
<td>threshold on number of generated cuts below which the ordinary separation is started</td>
<td>50</td>
</tr>
<tr>
<td>separating/closecuts/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/expbackoff</td>
<td>base for exponential increase of frequency at which separator (&lt;\text{closecuts}&gt;) is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/closecuts/inclobjcutoff</td>
<td>include an objective cutoff when computing the relative interior?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator (&lt;\text{closecuts}&gt;) ((0.0: \text{only on current best node}, 1.0: \text{on all nodes}))</td>
<td>1</td>
</tr>
<tr>
<td>separating/closecuts/maxlpiterfactor</td>
<td>factor for maximal LP iterations in relative interior computation compared to node LP iterations (negative for no limit)</td>
<td>10</td>
</tr>
<tr>
<td>separating/closecuts/maxunsuccessful</td>
<td>turn off separation in current node after unsuccessful calls ((-1 \text{ never turn off}))</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/priority</td>
<td>priority of separator (&lt;\text{closecuts}&gt;)</td>
<td>1000000</td>
</tr>
<tr>
<td>separating/closecuts/recompute relint</td>
<td>recompute relative interior point in each separation call?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/sepacombvalue</td>
<td>convex combination value for close cuts</td>
<td>0.3</td>
</tr>
<tr>
<td>separating/closecuts/separelrelint</td>
<td>generate close cuts w.r.t. relative interior point (best solution otherwise)?</td>
<td>1</td>
</tr>
</tbody>
</table>

5.36.4.231 separating/cmir

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/cmir/freq</td>
<td>frequency for calling separator (&lt;\text{cmir}&gt;) ((-1: \text{never}, 0: \text{only in root node}))</td>
<td>10</td>
</tr>
<tr>
<td>separating/cmir/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/expbackoff</td>
<td>base for exponential increase of frequency at which separator (&lt;\text{cmir}&gt;)\ is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/cmir/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;cmir&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/priority</td>
<td>priority of separator &lt;cmir&gt;</td>
<td>-100000</td>
</tr>
</tbody>
</table>

5.36.4.232 separating/convexproj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/convexproj/freq</td>
<td>frequency for calling separator &lt;convexproj&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/convexproj/maxdepth</td>
<td>maximal depth at which the separator is applied (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/convexproj/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/convexproj/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;convexproj&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/convexproj/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;convexproj&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/convexproj/nlpirerlimit</td>
<td>iteration limit of NLP solver; 0 for no limit</td>
<td>250</td>
</tr>
<tr>
<td>separating/convexproj/priority</td>
<td>priority of separator &lt;convexproj&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

Options for expert users

5.36.4.233 separating/disjunctive

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/disjunctive/freq</td>
<td>frequency for calling separator &lt;disjunctive&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/disjunctive/maxrank</td>
<td>maximal rank of a disj. cut that could not be scaled to integral coefficients (-1: unlimited)</td>
<td>20</td>
</tr>
<tr>
<td>separating/disjunctive/maxrankintegral</td>
<td>maximal rank of a disj. cut that could be scaled to integral coefficients (-1: unlimited)</td>
<td>-1</td>
</tr>
</tbody>
</table>

Options for expert users
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/disjunctive/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/disjunctive/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;disjunctive&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/disjunctive/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;disjunctive&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/disjunctive/maxconfsdelay</td>
<td>delay separation if number of conflict graph edges is larger than predefined value (-1: no limit)</td>
<td>1000000</td>
</tr>
<tr>
<td>separating/disjunctive/maxdepth</td>
<td>node depth of separating bipartite disjunctive cuts (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/disjunctive/maxinvcuts</td>
<td>maximal number of cuts investigated per iteration in a branching node</td>
<td>50</td>
</tr>
<tr>
<td>separating/disjunctive/maxinvcutsroot</td>
<td>maximal number of cuts investigated per iteration in the root node</td>
<td>250</td>
</tr>
<tr>
<td>separating/disjunctive/maxrounds</td>
<td>maximal number of separation rounds per iteration in a branching node (-1: no limit)</td>
<td>25</td>
</tr>
<tr>
<td>separating/disjunctive/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: no limit)</td>
<td>100</td>
</tr>
<tr>
<td>separating/disjunctive/maxweightrange</td>
<td>maximal valid range max(</td>
<td>weights</td>
</tr>
<tr>
<td>separating/disjunctive/priority</td>
<td>priority of separator &lt;disjunctive&gt;</td>
<td>10</td>
</tr>
<tr>
<td>separating/disjunctive/strengthen</td>
<td>strengthen cut if integer variables are present.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.234 separating/eccuts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/eccuts/cutmaxrange</td>
<td>maximal coef. range of a cut (max coef. divided by min coef.) in order to be added to LP relaxation</td>
<td>1e+07</td>
</tr>
<tr>
<td>separating/eccuts/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/eccuts/freq</td>
<td>frequency for calling separator <code>&lt;eccuts&gt;</code> (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxdepth</td>
<td>maximal depth at which the separator is applied (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxrounds</td>
<td>maximal number of eccuts separation rounds per node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxroundsroot</td>
<td>maximal number of eccuts separation rounds in the root node (-1: unlimited)</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxsepacuts</td>
<td>maximal number of edge-concave cuts separated per separation round</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxsepacutsroot</td>
<td>maximal number of edge-concave cuts separated per separation round in the root node</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/minviolation</td>
<td>minimal violation of an edge-concave cut to be separated</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 0.5]</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/delay</td>
<td>base for exponential increase of frequency at which separator <code>&lt;eccuts&gt;</code> is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/expbackoff</td>
<td>search for edge-concave aggregations of at most this size</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {3, ..., 5}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxaggrsize</td>
<td>maximum number of bilinear terms allowed to be in a quadratic constraint</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxbilinterms</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <code>&lt;eccuts&gt;</code> (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/maxstallrounds</td>
<td>maximum number of unsuccessful rounds in the edge-concave aggregation search</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/minaggrsize</td>
<td>search for edge-concave aggregations of at least this size</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: {3, ..., 5}</td>
<td></td>
</tr>
<tr>
<td>separating/eccuts/priority</td>
<td>priority of separator <code>&lt;eccuts&gt;</code></td>
<td>-13000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.235 separating/flowcover
### 5.36.4.236 separating/gauge

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/gauge/freq</td>
<td>frequency for calling separator &lt;gauge&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>
| Options for expert users
| separating/gauge/delay| should separator be delayed, if other separators found cuts?               | 0       |
|                      | Range: boolean                                                              |         |
| separating/gauge/expbackoff| base for exponential increase of frequency at which separator <gauge> is called (1: call at each multiple of frequency) | 4       |
|                      | Range: {1, ..., 100}                                                        |         |
| separating/gauge/maxbounddist| maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gauge> (0.0: only on current best node, 1.0: on all nodes) | 0       |
|                      | Range: [0, 1]                                                               |         |
| separating/gauge/priority| priority of separator <gauge>                                              | -100000|
|                      | Range: {-536870912, ..., 536870911}                                        |         |

### 5.36.4.237 separating/gomory

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/gomory/away</td>
<td>minimal integrality violation of a basis variable in order to try Gomory cut</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0.0001, 0.5]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/gomory/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/freq</td>
<td>frequency for calling separator &lt;gomory&gt; (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td>separating/gomory/maxrank</td>
<td>maximal rank of a gomory cut that could not be scaled to integral coefficients (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/gomory/maxrankintegral</td>
<td>maximal rank of a gomory cut that could be scaled to integral coefficients (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/gomory/maxrounds</td>
<td>maximal number of gomory separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/gomory/maxroundsroot</td>
<td>maximal number of gomory separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>separating/gomory/maxseapcuts</td>
<td>maximal number of gomory cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>separating/gomory/maxseapcutoffset</td>
<td>maximal number of gomory cuts separated per separation round in the root node</td>
<td>200</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/gomory/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/gomory/delayedcuts</td>
<td>should cuts be added to the delayed cut pool?</td>
<td>0</td>
</tr>
<tr>
<td>separating/gomory/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;gomory&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/gomory/forcecuts</td>
<td>if conversion to integral coefficients failed still consider the cut</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/genbothgomseg</td>
<td>Should both Gomory and strong CG cuts be generated (otherwise take best)?</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/makeintegral</td>
<td>try to scale cuts to integral coefficients</td>
<td>0</td>
</tr>
<tr>
<td>separating/gomory/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;gomory&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/priority</td>
<td>priority of separator &lt;gomory&gt;</td>
<td>-1000</td>
</tr>
<tr>
<td>separating/gomory/separaterows</td>
<td>separate rows with integral slack</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/gomory/sidetypebasis</td>
<td>choose side types of row (lhs/rhs) based on basis information?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/gomory/trysimplecg</td>
<td>try to generate strengthened Chvatal-Gomory cuts?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.238 separating/gomorymi

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/gomorymi/freq</td>
<td>frequency for calling separator <code>&lt;gomorymi&gt;</code> (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/gomorymi/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/gomorymi/expbackoff</td>
<td>base for exponential increase of frequency at which separator <code>&lt;gomorymi&gt;</code> is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/gomorymi/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <code>&lt;gomorymi&gt;</code> (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/gomorymi/priority</td>
<td>priority of separator <code>&lt;gomorymi&gt;</code></td>
<td>-100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.239 separating/impliedbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/impliedbounds/freq</td>
<td>frequency for calling separator <code>&lt;impliedbounds&gt;</code> (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/impliedbounds/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/impliedbounds/expbackoff</td>
<td>base for exponential increase of frequency at which separator <code>&lt;impliedbounds&gt;</code> is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/impliedbounds/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <code>&lt;impliedbounds&gt;</code> (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/impliedbounds/priority</td>
<td>priority of separator &lt;impliedbounds&gt;</td>
<td>-50</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>separating/impliedbounds/usetwosizecliques</td>
<td>should violated inequalities for cliques with 2 variables be separated?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.240 separating/interminor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/interminor/freq</td>
<td>frequency for calling separator &lt;interminor&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/mincutviol</td>
<td>minimum required violation of a cut</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/usebounds</td>
<td>whether to also enforce nonnegativity bounds of principle minors</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/usestrengthening</td>
<td>whether to use strengthened intersection cuts to separate minors</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/interminor/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;interminor&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/maxboundddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;interminor&gt; (0.0: on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/interminor/priority</td>
<td>priority of separator &lt;interminor&gt;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.241 separating/intobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/intobj/freq</td>
<td>frequency for calling separator &lt;intobj&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>
### Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/intobj/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/intobj/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;intobj&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/intobj/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;intobj&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/intobj/priority</td>
<td>priority of separator &lt;intobj&gt;</td>
<td>-100</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.242 separating/knapsackcover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/knapsackcover/freq</td>
<td>frequency for calling separator &lt;knapsackcover&gt; (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/knapsackcover/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/knapsackcover/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;knapsackcover&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/knapsackcover/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;knapsackcover&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/knapsackcover/priority</td>
<td>priority of separator &lt;knapsackcover&gt;</td>
<td>-100000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.243 separating/mcf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/mcf/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/freq</td>
<td>frequency for calling separator &lt;mcf&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/mcf/maxsepacuts</td>
<td>maximal number of mcf cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxsepacutsroot</td>
<td>maximal number of mcf cuts separated per separation round in the root node – default separation</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/mcf/checkcutshoreconnectivity</td>
<td>should we separate only if the cuts shores are connected?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;mcf&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/fixintegralrhs</td>
<td>should an additional variable be complemented if f0 = 0?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxarcinconsistencyratio</td>
<td>maximum inconsistency ratio of arcs not to be deleted</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node's dual bound for applying separator &lt;mcf&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxinconsistencyratio</td>
<td>maximum inconsistency ratio for separation at all</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxtestdelta</td>
<td>maximal number of different deltas to try (-1: unlimited) – default separation</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/maxweightrange</td>
<td>maximal valid range max(</td>
<td>weights</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/modeltype</td>
<td>model type of network (0: auto, 1:directed, 2:undirected)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/nclusters</td>
<td>number of clusters to generate in the shrunken network – default separation</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: {2, ..., 32}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/priority</td>
<td>priority of separator &lt;mcf&gt;</td>
<td>-10000</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>separating/mcf/separateflowcutset</td>
<td>should we separate flowcutset inequalities on the network cuts?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/mcf/separateknapsack</td>
<td>should we separate knapsack cover inequalities on the network cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/separatesinglenodecuts</td>
<td>should we separate inequalities based on single-node cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/trynegscaling</td>
<td>should negative values also be tested in scaling?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.244 separating/minor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/minor/freq</td>
<td>frequency for calling separator &lt;minor&gt; (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td>separating/minor/ignorepackingconss</td>
<td>whether to ignore circle packing constraints during minor detection</td>
<td>1</td>
</tr>
<tr>
<td>separating/minor/maxminorsconst</td>
<td>constant for the maximum number of minors, i.e., max(const, fac * # quadratic terms)</td>
<td>3000</td>
</tr>
<tr>
<td>separating/minor/maxminorsfac</td>
<td>factor for the maximum number of minors, i.e., max(const, fac * # quadratic terms)</td>
<td>10</td>
</tr>
<tr>
<td>separating/minor/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>separating/minor/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/minor/mincutviol</td>
<td>minimum required violation of a cut</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/minor/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/minor/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;minor&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/minor/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;minor&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/minor/priority</td>
<td>priority of separator &lt;minor&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.245 separating/mixing
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/mixing/freq</td>
<td>frequency for calling separator &lt;mixing&gt; (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/maxmuns</td>
<td>maximal number of consecutive unsuccessful iterations</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/maxrounds</td>
<td>maximal number of mixing separation rounds per node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/maxroundsroot</td>
<td>maximal number of mixing separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/mixing/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;mixing&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 100}</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/iscutsonints</td>
<td>Should general integer variables be used to generate cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;mixing&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/priority</td>
<td>priority of separator &lt;mixing&gt;</td>
<td>-50</td>
</tr>
<tr>
<td></td>
<td>Range: {-536870912, ..., 536870911}</td>
<td></td>
</tr>
<tr>
<td>separating/mixing/uselocalbounds</td>
<td>Should local bounds be used?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.246 separating/oddcycle

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/oddcycle/freq</td>
<td>frequency for calling separator &lt;oddcycle&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., 65534}</td>
<td></td>
</tr>
<tr>
<td>separating/oddcycle/liftoddcycles</td>
<td>Should odd cycle cuts be lifted?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/oddcycle/maxrounds</td>
<td>maximal number of oddcycle separation rounds per node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/oddcycle/maxroundsroot</td>
<td>maximal number of oddcycle separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {-1, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>separating/oddcycle/maxsepacuts</td>
<td>maximal number of oddcycle cuts separated per separation round</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, ..., ∞}</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>separating/oddcycle/maxsepacutsroot</td>
<td>maximal number of oddcycle cuts separated per separation round in the root node</td>
<td>5000</td>
</tr>
<tr>
<td>separating/oddcycle/usegls</td>
<td>Should the search method by Groetschel, Lovasz, Schrijver be used? Otherwise use levelgraph method by Hoffman, Padberg.</td>
<td>1</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/oddcycle/addselfarcs</td>
<td>add links between a variable and its negated</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/allowmultiplecuts</td>
<td>Even if a variable is already covered by a cut, still allow another cut to cover it too?</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/cutthreshold</td>
<td>maximal number of other cuts s.t. separation is applied (-1 for direct call)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/oddcycle/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;oddcycle&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/oddcycle/includetriangles</td>
<td>separate triangles found as 3-cycles or repaired larger cycles</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/lpliftcoef</td>
<td>Choose lifting candidate by coef*lpvalue or only by coef?</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;oddcycle&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/maxcutslevel</td>
<td>maximal number of oddcycle cuts generated in every level of the level graph</td>
<td>50</td>
</tr>
<tr>
<td>separating/oddcycle/maxcutsroot</td>
<td>maximal number of oddcycle cuts generated per chosen variable as root of the level graph</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/maxnlevels</td>
<td>maximal number of levels in level graph</td>
<td>20</td>
</tr>
<tr>
<td>separating/oddcycle/maxpernodeslevel</td>
<td>percentage of nodes allowed in the same level of the level graph [0-100]</td>
<td>100</td>
</tr>
<tr>
<td>separating/oddcycle/maxreference</td>
<td>minimal weight on an edge (in level graph or bipartite graph)</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/maxunsuccessfull</td>
<td>number of unsuccessful calls at current node</td>
<td>3</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/oddcycle/multiplecuts</td>
<td>Even if a variable is already covered by a cut, still try it as start node for a cycle search? Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/offsetnodeslevel</td>
<td>offset of nodes allowed in the same level of the level graph (additional to the percentage of lev-nodes) Range: {0, ..., ∞}</td>
<td>10</td>
</tr>
<tr>
<td>separating/oddcycle/offsettestvars</td>
<td>offset of variables to try the chosen method on (additional to the percentage of testvars) Range: {0, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>separating/oddcycle/percenttestvars</td>
<td>percentage of variables to try the chosen method on [0-100] Range: {0, ..., 100}</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/priority</td>
<td>priority of separator &lt;oddcycle&gt; Range: {-536870912, ..., 536870911}</td>
<td>-15000</td>
</tr>
<tr>
<td>separating/oddcycle/recalcliftcoef</td>
<td>Calculate lifting coefficient of every candidate in every step (or only if its chosen)? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/repaircycles</td>
<td>try to repair violated cycles with double appearance of a variable Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/scalingfactor</td>
<td>factor for scaling of the arc-weights Range: {1, ..., ∞}</td>
<td>1000</td>
</tr>
<tr>
<td>separating/oddcycle/sortrootneighbors</td>
<td>sort level of the root neighbors by fractionality (maxfrac) Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/sortswitch</td>
<td>use sorted variable array (unsorted(0), maxlp(1), minlp(2), maxfrac(3), minfrac(4)) Range: {0, ..., 4}</td>
<td>3</td>
</tr>
</tbody>
</table>

5.36.4.247 separating/rapidlearning

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/rapidlearning/freq</td>
<td>frequency for calling separator &lt;rapidlearning&gt; (-1: never, 0: only in root node) Range: {-1, ..., 65534}</td>
<td>5</td>
</tr>
</tbody>
</table>

Options for expert users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/rapidlearning/applybdchgs</td>
<td>should the found global bound deductions be applied in the original SCIP? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/applyconflicts</td>
<td>should the found conflicts be applied in the original SCIP? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/applyinervals</td>
<td>should the inference values be used as initialization in the original SCIP? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/applyprimalsol</td>
<td>should the incumbent solution be copied to the original SCIP? Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/rapidlearning/aplysolved</td>
<td>should a solved status be copied to the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/checkdegeneracy</td>
<td>should local LP degeneracy be checked?</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/checkdualbound</td>
<td>should the progress on the dual bound be checked?</td>
<td>0</td>
</tr>
<tr>
<td>separating/rapidlearning/checkexec</td>
<td>check whether rapid learning should be executed</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/checkleaves</td>
<td>should the ratio of leaves proven to be infeasible and exceeding the cutoff bound be checked?</td>
<td>0</td>
</tr>
<tr>
<td>separating/rapidlearning/checknsols</td>
<td>should the number of solutions found so far be checked?</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/checkobj</td>
<td>should the (local) objective function be checked?</td>
<td>0</td>
</tr>
<tr>
<td>separating/rapidlearning/contvars</td>
<td>should rapid learning be applied when there are continuous variables?</td>
<td>0</td>
</tr>
<tr>
<td>separating/rapidlearning/contvarsquot</td>
<td>maximal portion of continuous variables to apply rapid learning</td>
<td>0.3</td>
</tr>
<tr>
<td>separating/rapidlearning/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/rapidlearning/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;rapidlearning&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/rapidlearning/lpiterquot</td>
<td>maximal fraction of LP iterations compared to node LP iterations</td>
<td>0.2</td>
</tr>
<tr>
<td>separating/rapidlearning/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;rapidlearning&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/rapidlearning/maxcalls</td>
<td>maximum number of overall calls</td>
<td>100</td>
</tr>
<tr>
<td>separating/rapidlearning/maxnconss</td>
<td>maximum problem size (constraints) for which rapid learning will be called</td>
<td>10000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/rapidlearning/maxnodes</td>
<td>maximum number of nodes considered in rapid learning run</td>
<td>5000</td>
</tr>
<tr>
<td>separating/rapidlearning/maxnvars</td>
<td>maximum problem size (variables) for which rapid learning will be called</td>
<td>10000</td>
</tr>
<tr>
<td>separating/rapidlearning/mindegeneracy</td>
<td>minimal degeneracy threshold to allow local rapid learning</td>
<td>0.7</td>
</tr>
<tr>
<td>separating/rapidlearning/mininflpratio</td>
<td>minimal threshold of inf/obj leaves to allow local rapid learning</td>
<td>10</td>
</tr>
<tr>
<td>separating/rapidlearning/minnodes</td>
<td>minimum number of nodes considered in rapid learning run</td>
<td>500</td>
</tr>
<tr>
<td>separating/rapidlearning/minvarconsratio</td>
<td>minimal ratio of unfixed variables in relation to basis size to allow local rapid learning</td>
<td>2</td>
</tr>
<tr>
<td>separating/rapidlearning/nwaitingnodes</td>
<td>number of nodes that should be processed before rapid learning is executed locally based on the progress of the dualbound</td>
<td>100</td>
</tr>
<tr>
<td>separating/rapidlearning/priority</td>
<td>priority of separator &lt;rapidlearning&gt;</td>
<td>-1200000</td>
</tr>
<tr>
<td>separating/rapidlearning/reducedinfer</td>
<td>should the inference values only be used when rapidlearning found other reductions?</td>
<td>0</td>
</tr>
</tbody>
</table>

5.36.4.248 separating/rlt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/rlt/addtopool</td>
<td>if set to true, globally valid RLT cuts are added to the global cut pool</td>
<td>1</td>
</tr>
<tr>
<td>separating/rlt/detecthidden</td>
<td>if set to true, hidden products are detected and separated by McCormick cuts</td>
<td>0</td>
</tr>
<tr>
<td>separating/rlt/freq</td>
<td>frequency for calling separator &lt;rlt&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/rlt/hiddenrlt</td>
<td>whether RLT cuts (TRUE) or only McCormick inequalities (FALSE) should be added for hidden products</td>
<td>0</td>
</tr>
<tr>
<td>separating/rlt/maxncuts</td>
<td>maximal number of rlt-cuts that are added per round (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/rlt/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/rlt/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: (-1, \ldots, \infty)</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/maxunknownterms</td>
<td>maximal number of unknown bilinear terms a row is still used with (-1: unlimited)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: (-1, \ldots, \infty)</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/maxusedvars</td>
<td>maximal number of variables used to compute rlt cuts (-1: unlimited)</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: (-1, \ldots, \infty)</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/onlycontrows</td>
<td>if set to true, only continuous rows are used for rlt cuts</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/onlyeqrows</td>
<td>if set to true, only equality rows are used for rlt cuts</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/onlyoriginal</td>
<td>if set to true, only original rows and variables are used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/useinsubscip</td>
<td>if set to true, rlt is also used in sub-scips</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/useprojection</td>
<td>if set to true, projected rows are checked first</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Options for expert users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/rlt/badscore</td>
<td>threshold for score of cut relative to best score to be discarded</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/dircutoffdistweight</td>
<td>weight of directed cutoff distance in cut score calculation</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/efficacyweight</td>
<td>weight of efficacy in cut score calculation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/expbackoff</td>
<td>base for exponential increase of frequency at which separator (&lt;\text{rlt}&gt;) is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Range: ([1, \ldots, 100])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/goodmaxparall</td>
<td>maximum parallelism for good cuts</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/goodscore</td>
<td>threshold for score of cut relative to best score to be considered good, so that less strict filtering is applied</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator (&lt;\text{rlt}&gt;) (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/maxparall</td>
<td>maximum parallelism for non-good cuts</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>separating/rlt/objparalweight</td>
<td>weight of objective parallelism in cut score calculation</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/rlt/priority</td>
<td>priority of separator <code>&lt;rlt&gt;</code>&lt;br&gt;Range: <code>{−536870912, ..., 536870911}</code></td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.36.4.249 separating/strongcg

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/strongcg/freq</td>
<td>frequency for calling separator <code>&lt;strongcg&gt;</code>&lt;br&gt;(-1: never, 0: only in root node)&lt;br&gt;Range: <code>{−1, ..., 65534}</code></td>
<td>10</td>
</tr>
<tr>
<td>Options for expert users</td>
<td>should separator be delayed, if other separators found cuts?&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
<tr>
<td>separating/strongcg/delay</td>
<td>base for exponential increase of frequency at which separator <code>&lt;strongcg&gt;</code> is called (1: call at each multiple of frequency)&lt;br&gt;Range: <code>{1, ..., 100}</code></td>
<td>4</td>
</tr>
<tr>
<td>separating/strongcg/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <code>&lt;strongcg&gt;</code>&lt;br&gt;Range: <code>{0.0: only on current best node, 1.0: on all nodes}</code></td>
<td>0</td>
</tr>
<tr>
<td>separating/strongcg/priority</td>
<td>priority of separator <code>&lt;strongcg&gt;</code>&lt;br&gt;Range: <code>{−536870912, ..., 536870911}</code></td>
<td>-100000</td>
</tr>
</tbody>
</table>

### 5.36.4.250 separating/zerohalf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/zerohalf/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/freq</td>
<td>frequency for calling separator <code>&lt;zerohalf&gt;</code>&lt;br&gt;(-1: never, 0: only in root node)&lt;br&gt;Range: <code>{−1, ..., 65534}</code></td>
<td>10</td>
</tr>
<tr>
<td>separating/zerohalf/initseed</td>
<td>initial seed used for random tie-breaking in cut selection&lt;br&gt;Range: <code>{0, ..., ∞}</code></td>
<td>24301</td>
</tr>
<tr>
<td>separating/zerohalf/maxcutcands</td>
<td>maximal number of zerohalf cuts considered per separation round&lt;br&gt;Range: <code>{0, ..., ∞}</code></td>
<td>2000</td>
</tr>
<tr>
<td>separating/zerohalf/maxrounds</td>
<td>maximal number of zerohalf separation rounds per node (-1: unlimited)&lt;br&gt;Range: <code>{−1, ..., ∞}</code></td>
<td>5</td>
</tr>
<tr>
<td>separating/zerohalf/maxroundsroot</td>
<td>maximal number of zerohalf separation rounds in the root node (-1: unlimited)&lt;br&gt;Range: <code>{−1, ..., ∞}</code></td>
<td>20</td>
</tr>
<tr>
<td>separating/zerohalf/maxsepacuts</td>
<td>maximal number of zerohalf cuts separated per separation round&lt;br&gt;Range: <code>{0, ..., ∞}</code></td>
<td>20</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/zerohalf/maxsepacutsroot</td>
<td>maximal number of zerohalf cuts separated per separation round in the root node</td>
<td>100</td>
</tr>
<tr>
<td>Options for expert users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>separating/zerohalf/badscore</td>
<td>threshold for score of cut relative to best score to be discarded</td>
<td>0.5</td>
</tr>
<tr>
<td>separating/zerohalf/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/densityoffset</td>
<td>additional number of variables allowed in row on top of density</td>
<td>100</td>
</tr>
<tr>
<td>separating/zerohalf/dircutoffdistweight</td>
<td>weight of directed cutoff distance in cut score calculation</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/efficacyweight</td>
<td>weight of efficacy in cut score calculation</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/expbackoff</td>
<td>base for exponential increase of frequency at which separator &lt;zerohalf&gt; is called (1: call at each multiple of frequency)</td>
<td>4</td>
</tr>
<tr>
<td>separating/zerohalf/goodmaxparall</td>
<td>maximum parallelism for good cuts</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/zerohalf/goodscore</td>
<td>threshold for score of cut relative to best score to be considered good, so that less strict filtering is applied</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/maxbounddist</td>
<td>maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator &lt;zerohalf&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/maxparall</td>
<td>maximum parallelism for non-good cuts</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/zerohalf/maxrowdensity</td>
<td>maximal density of row to be used in aggregation</td>
<td>0.05</td>
</tr>
<tr>
<td>separating/zerohalf/maxslack</td>
<td>maximal slack of rows to be used in aggregation</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/maxslackroot</td>
<td>maximal slack of rows to be used in aggregation in the root node</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/minviol</td>
<td>minimal violation to generate zerohalfcut</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/zerohalf/objparalweight</td>
<td>weight of objective parallelism in cut score calculation</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/priority</td>
<td>priority of separator &lt;zerohalf&gt;</td>
<td>-6000</td>
</tr>
</tbody>
</table>
5.36.4.251 solvingphases

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>solvingphases/enabled</td>
<td>should the event handler adapt the solver behavior?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/fallback</td>
<td>should the event handler fall back from optimal phase?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/feassetname</td>
<td>settings file for feasibility phase – precedence over emphasis settings</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>solvingphases/improvesetname</td>
<td>settings file for improvement phase – precedence over emphasis settings</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>solvingphases/interruptoptimal</td>
<td>should the event handler interrupt the solving process after optimal solution was found?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/nodeoffset</td>
<td>node offset for rank-1 and estimate transitions</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., \infty}</td>
<td></td>
</tr>
<tr>
<td>solvingphases/optimalvalue</td>
<td>optimal solution value for problem</td>
<td>\infty</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>solvingphases/proofsetname</td>
<td>settings file for proof phase – precedence over emphasis settings</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>solvingphases/testmode</td>
<td>should the event handler test all phase transitions?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/transitionmethod</td>
<td>transition method: Possible options are 'e'estimate,'l'ogarithmic regression,'o'ptimal-value based,'r'ank-1</td>
<td>r</td>
</tr>
<tr>
<td></td>
<td>Range: e, l, o, r</td>
<td></td>
</tr>
<tr>
<td>solvingphases/useemphsettings</td>
<td>should emphasis settings for the solving phases be used, or settings files?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/userrestart1to2</td>
<td>should a restart be applied between the feasibility and improvement phase?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/userrestart2to3</td>
<td>should a restart be applied between the improvement and the proof phase?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>solvingphases/xtype</td>
<td>x-type for logarithmic regression - (t)ime, (n)odes, (l)p iterations</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>Range: l, n, t</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.252 table/branchrules

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/branchrules/active</td>
<td>is statistics table &lt;branchrules&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.253 table/compression
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/compression/active</td>
<td>is statistics table &lt;compression&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.254 table/concurrentsolver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/concurrentsolver/active</td>
<td>is statistics table &lt;concurrentsolver&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.255 table/conflict

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/conflict/active</td>
<td>is statistics table &lt;conflict&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.256 table/cons_nonlinear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/cons_nonlinear/active</td>
<td>is statistics table &lt;cons_nonlinear&gt; active&lt;br&gt;Range: boolean</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.36.4.257 table/constiming

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/constiming/active</td>
<td>is statistics table &lt;constiming&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.258 table/constraint

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/constraint/active</td>
<td>is statistics table &lt;constraint&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.259 table/cutsel

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/cutsel/active</td>
<td>is statistics table &lt;cutsel&gt; active&lt;br&gt;Range: boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.36.4.260 table/estim
### 5.36.4.261 table/exprhdlr

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/exprhdlr/active</td>
<td>is statistics table &lt;exprhdlr&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.262 table/heuristics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/heuristics/active</td>
<td>is statistics table &lt;heuristics&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.263 table/lp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/lp/active</td>
<td>is statistics table &lt;lp&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.264 table/neighborhood

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/neighborhood/active</td>
<td>is statistics table &lt;neighborhood&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.265 table/nlhdlr

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/nlhdlr/active</td>
<td>is statistics table &lt;nlhdlr&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.266 table/nlhdlr_bilinear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/nlhdlr_bilinear/active</td>
<td>is statistics table &lt;nlhdlr_bilinear&gt; active</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.267 table/nlhdlr_quadratic

---
### 5.36.4.268 table/nlp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/nlp/active</td>
<td>is statistics table &lt;nlp&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.269 table/nlpi

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/nlpi/active</td>
<td>is statistics table &lt;nlpi&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.270 table/orbitalfixing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/orbitalfixing/active</td>
<td>is statistics table &lt;orbitalfixing&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.271 table/origprob

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/origprob/active</td>
<td>is statistics table &lt;origprob&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.272 table/presolvedprob

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/presolvedprob/active</td>
<td>is statistics table &lt;resolvedprob&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.273 table/presolver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/presolver/active</td>
<td>is statistics table &lt;presolver&gt; active</td>
<td>1</td>
</tr>
<tr>
<td>Range: boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.36.4.274 table/pricer
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/pricer/active</td>
<td>is statistics table &lt;pricer&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.275 table/propagator

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/propagator/active</td>
<td>is statistics table &lt;propagator&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.276 table/relaxator

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/relaxator/active</td>
<td>is statistics table &lt;relaxator&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.277 table/root

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/root/active</td>
<td>is statistics table &lt;root&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.278 table/separator

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/separator/active</td>
<td>is statistics table &lt;separator&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.279 table/solution

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/solution/active</td>
<td>is statistics table &lt;solution&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.280 table/status

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/status/active</td>
<td>is statistics table &lt;status&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

5.36.4.281 table/timing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Option Description Default

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/timing/active</td>
<td>is statistics table &lt;timing&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.282 table/tree

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>table/tree/active</td>
<td>is statistics table &lt;tree&gt; active</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.36.4.283 timing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>timing/clocktype</td>
<td>default clock type (1: CPU user seconds, 2: wall clock time)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: {1, ..., 2}</td>
<td></td>
</tr>
<tr>
<td>timing/enabled</td>
<td>is timing enabled?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>timing/nlpieval</td>
<td>should time for evaluation in NLP solves be measured?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>timing/rareclockcheck</td>
<td>should clock checks of solving time be performed less frequently</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(note: time limit could be exceeded slightly)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>timing/reading</td>
<td>belongs reading time to solving time?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>timing/statistic.timing</td>
<td>should timing for statistic output be performed?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.37 SELKIE

#### Authors

Youngdae Kim, youngdae@anl.gov; Mathematics and Computer Science Division; Argonne National Laboratory  
Michael C. Ferris; Computer Science Department; UW-Madison

#### Date

23 Jan 2019

#### 5.37.1 Introduction

This document describes the GAMS/SELKIE solver for equilibrium models. This is a research code, so you should set your expectations accordingly. It doesn't come with extensive documentation or a history of continuous development and improvement. To learn more about the algorithms SELKIE implements and related topics, see Y. Kim's Ph.D. thesis or home page.
5.37.2 Usage

To explicitly request that a model be solved with SELKIE, insert the statement

```
option EMP = selkie;
```

somewhere before the solve statement in the GAMS source. If SELKIE has been selected as the default EMP solver (e.g. by including `EMP=selkie` in the command line arguments of the GAMS run) the above statement is not necessary.

SELKIE-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

An example of a valid SELKIE option file is:

```
* lines starting with a star are comments
diagonalization_method jacobi
parallel_jacobi yes
output_group_log yes
```

5.37.3 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_group</td>
<td>groups of agents</td>
<td>empty</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>convergence tolerance of residual</td>
<td>1e-6</td>
</tr>
<tr>
<td>coupling_constraints</td>
<td>names of coupling constraints</td>
<td>empty</td>
</tr>
<tr>
<td>coupling_variables</td>
<td>names of coupling variables</td>
<td>empty</td>
</tr>
<tr>
<td>cumulative_iteration_limit</td>
<td>maximum number of cumulative iterations</td>
<td>5000</td>
</tr>
<tr>
<td>damping_factor</td>
<td>damping factor b/w old and new values</td>
<td>1.0</td>
</tr>
<tr>
<td>dantzig_wolfe_initial_samples</td>
<td>initial number of samples for DW decomposition</td>
<td>100</td>
</tr>
<tr>
<td>dantzig_wolfe_master_subsolveropt</td>
<td>Subsolver option number for the master of DW</td>
<td></td>
</tr>
<tr>
<td>dantzig_wolfe_subproblem_subsolveropt</td>
<td>Subsolver option number for the sub of DW</td>
<td></td>
</tr>
<tr>
<td>decomposition_method</td>
<td>decomposition method to apply</td>
<td>diagonalization</td>
</tr>
<tr>
<td></td>
<td>diagonalization: Diagonalization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dantzig_wolfe: Dantzig-Wolfe</td>
<td></td>
</tr>
<tr>
<td>delay_residual_check</td>
<td>delay residual checking</td>
<td>0</td>
</tr>
<tr>
<td>detect_independent_groups</td>
<td>detect independent groups of agents</td>
<td>0</td>
</tr>
<tr>
<td>deviation_agent_list</td>
<td>agents whose deviations are computed</td>
<td>empty</td>
</tr>
<tr>
<td>deviation_convergence_tolerance</td>
<td>convergence tolerance of deviation</td>
<td>1e-10</td>
</tr>
<tr>
<td>deviation_delay_tolerance</td>
<td>deviation tolerance until residual is not checked</td>
<td>1e-8</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>diagonalization_method</td>
<td>see the values below</td>
<td>gauss_seidel</td>
</tr>
<tr>
<td></td>
<td>gauss_seidel: Gauss-Seidel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gauss_seidel_random: Randomized Gauss-Seidel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gauss_seidel_random_sweep: Randomized Gauss-Seidel with sweep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gauss_southwell: Residual-based Gauss-Seidel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jacobi: Jacobi</td>
<td></td>
</tr>
<tr>
<td>epsilon</td>
<td>ignored if abs is less than this</td>
<td>1e-8</td>
</tr>
<tr>
<td>major_iteration_limit</td>
<td>maximum number of major iterations</td>
<td>100</td>
</tr>
<tr>
<td>minor_iteration_limit</td>
<td>maximum number of minor iterations</td>
<td>1000</td>
</tr>
<tr>
<td>nlp_solver</td>
<td>NLP solver</td>
<td>conopt</td>
</tr>
<tr>
<td></td>
<td>antigone: Antigone</td>
<td></td>
</tr>
<tr>
<td></td>
<td>baron: Baron</td>
<td></td>
</tr>
<tr>
<td></td>
<td>conopt: Conopt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>conopt4: Conopt4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipopt: Ipopt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>knitro: Knitro</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lindo: Lindo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lindoglobal: Lindoglobal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>minos: Minos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mosek: Mosek</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scip: Scip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>snopt: Snopt</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>whether to output logs</td>
<td>1</td>
</tr>
<tr>
<td>output_group_log</td>
<td>whether to output group logs</td>
<td>0</td>
</tr>
<tr>
<td>output_iteration_log</td>
<td>whether to output iteration logs</td>
<td>1</td>
</tr>
<tr>
<td>output_subsolver_log</td>
<td>whether to output subsolver logs</td>
<td>0</td>
</tr>
<tr>
<td>parallel_agent_generation</td>
<td>generate agents problems in parallel</td>
<td>0</td>
</tr>
<tr>
<td>parallel_jacobi</td>
<td>run jacobi in parallel</td>
<td>0</td>
</tr>
<tr>
<td>proximal_increase_factor</td>
<td>increasing factor for proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_initial_maximum</td>
<td>initial maximum value of proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_maximum</td>
<td>maximum of proximal term</td>
<td></td>
</tr>
<tr>
<td>proximalMerit_factor</td>
<td>factor that is multiplied to residual</td>
<td></td>
</tr>
<tr>
<td>proximal_minimum</td>
<td>minimum value of proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_perturbation</td>
<td>penalty value of proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_perturbation_dual</td>
<td>penalty value of dual proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_perturbation_primal</td>
<td>penalty value of primal proximal term</td>
<td></td>
</tr>
<tr>
<td>proximal_residual_change</td>
<td>ratio of residual change</td>
<td></td>
</tr>
<tr>
<td>proximal_shrink_factor</td>
<td>shrinking factor for proximal term</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>proximal_use</td>
<td>determine the proximal mode</td>
<td>empty</td>
</tr>
<tr>
<td>record_ownership_within_group</td>
<td>record ownership of agents within group</td>
<td>0</td>
</tr>
<tr>
<td>remove_isolated_implvar</td>
<td>omit isolated implicit variables</td>
<td>1</td>
</tr>
<tr>
<td>shared_equation</td>
<td>allow shared equation</td>
<td>0</td>
</tr>
<tr>
<td>stop_on_subsolver_failure</td>
<td>stop selkie if subsolver fails to solve</td>
<td>1</td>
</tr>
<tr>
<td>subsolveropt</td>
<td>subsolver option file number</td>
<td>1</td>
</tr>
<tr>
<td>time_limit</td>
<td>maximum running time in seconds</td>
<td>3600</td>
</tr>
<tr>
<td>use_mcpopt</td>
<td>use MCP to solve optimization agent</td>
<td>0</td>
</tr>
<tr>
<td>write_agent_model</td>
<td>write agent model into file</td>
<td>1</td>
</tr>
</tbody>
</table>

5.38 SHOT

SHOT (Supporting Hyperplane Optimization Toolkit) is a deterministic solver for mixed-integer nonlinear programming problems (MINLPs).

Originally, SHOT was intended for convex MINLP problems only, but now also has functionality to solve nonconvex MINLP problems as a heuristic method without providing any guarantees of global optimality. SHOT can solve certain nonconvex problem types to global optimality as well.

SHOT has mainly been developed by Andreas Lundell (Åbo Akademi University, Finland) and Jan Kronqvist (Imperial College London, UK). For more details, see [126] [122] [112] [113] [121].

SHOT supports GAMS equations that use the following intrinsic functions: abs, cos, cvPower, div, exp, log, log10, log2, pi, power, rPower, sin, sqr, sqrt, vcPower.

5.38.1 Algorithm

SHOT is based on iteratively creating a tighter polyhedral approximation of the nonlinear feasible set by generating supporting hyperplanes or cutting planes. These linearized problems are then solved with a mixed-integer linear programming (MIP) solver. GAMS/SHOT uses CPLEX, if a GAMS/CPLEX license is available, and otherwise CBC. Users with a license from Gurobi can also select Gurobi as MIP solver. If CPLEX or Gurobi is used, the subproblems can also include quadratic and bilinear nonlinearities directly.

The solution to the outer approximation problem provides a dual bound (i.e., a lower bound when solving a minimization problem) to the optimal value of the original problem if it is convex. If the problem is nonconvex, convergence to the global optimal solution cannot be guaranteed (but might be achieved for certain classes of problems, cf. [122]).

To get a primal bound (i.e., an upper bound when solving a minimization problem) on the optimal value, SHOT utilizes the following heuristics:

- Solving nonlinear programming (NLP) problems where the integer variables have been fixed to valid values. This is done by calling an NLP solver, which is either Ipopt, one of the GAMS NLP solvers, or SHOT itself.
5.38 SHOT

- By checking solutions from the MIP solver's solution pool for points that fulfill also the nonlinear constraints in the original MINLP problem.
- By performing root searches.

When a termination criterion like a tolerance on the relative or absolute objective gap or a time limit is fulfilled, SHOT terminates and returns the current primal solution to GAMS. If the original problem is convex and SHOT could close the objective gap, then this is a global optimal solution to the problem. If it is nonconvex, then there is normally no guarantee that such a solution can be found. However, SHOT will always, in addition to a primal solution, return a valid dual bound on the solution in model attribute objest, unless Model.Convexity.AssumeConvex has been enabled.

5.38.2 Usage

The following statement can be used inside your GAMS program to specify using SHOT

```gams
Option MINLP = SHOT; { or MIQCP }
```

The above statement should appear before the Solve statement. If SHOT was specified as the default MINLP or MIQCP solver during GAMS installation, the above statement is not necessary.

5.38.2.1 Specification of SHOT Options

GAMS/SHOT supports the GAMS parameters reslim, iterlim, nodlim, optcr, optca, cutoff, and threads.

Options can be specified by a SHOT options file. A SHOT options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs).

A small example for a shot.opt file is:

```gams
Dual.CutStrategy = 1
Dual.MIP.Solver = 2
Output.Console.DualSolver.Show = true
```

It causes GAMS/SHOT to use the Extended Cutting Plane (ECP) method instead of the Extended Supporting Hyperplane (EHP) method, changes the MIP solver to CBC, and enables showing the output of the solver that computes dual bounds (typically the MIP solver).

Attention

SHOT requires options to be specified using exactly the names as specified in the documentation. That is, also casing matters.

5.38.3 List of SHOT Options

In the following, we give a detailed list of all SHOT options.

5.38.3.1 Solver output

These settings control how much and what output is shown to the user from the solver.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output.Debug.Path</td>
<td>The folder where to save the debug information</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>Output.GAMS.AlternateSolutionsFile</td>
<td>Name of GAMS GDX file to write alternative solutions to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>Output.Console.Iteration.Detail</td>
<td>When should the fixed strategy be used</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Full</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: On objective gap update</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: On objective gap update and all primal NLP calls</td>
<td></td>
</tr>
<tr>
<td>Output.Console.LogLevel</td>
<td>Log level for console output</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0: Trace</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Debug</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Info</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Warning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: Error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5: Critical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6: Off</td>
<td></td>
</tr>
<tr>
<td>Output.Console.DualSolver.Show</td>
<td>Show output from dual solver on console</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Output.Console.PrimalSolver.Show</td>
<td>Show output from primal solver on console</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Output.Debug.Enable</td>
<td>Use debug functionality</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 5.38.3.2 Subsolver functionality

These settings allow for more direct control of the different subsolvers utilized in SHOT.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>Subsolver.GAMS.NLP.OptionsFilename</td>
<td>Options file for the NLP solver in GAMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>Subsolver.GAMS.NLP.Solver</td>
<td>NLP solver to use in GAMS (auto: SHOT chooses)</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>Range: string</td>
<td></td>
</tr>
<tr>
<td>Subsolver.Cbc.NodeStrategy</td>
<td>Node strategy</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0: depth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: downdepth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: downfewest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: fewest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: hybrid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5: updepth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6: upfewest</td>
<td></td>
</tr>
<tr>
<td>Subsolver.Cbc.Scaling</td>
<td>Whether to scale problem</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0: automatic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: dynamic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: equilibrium</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: geometric</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: off</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5: rowsonly</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| Subsolver.Cbc.Strategy       | This turns on newer features  
0: easy problems  
1: default  
2: aggressive                  | 1       |
| Subsolver.Cplex.FeasOptMode  | Strategy to use for the feasibility repair  
0: Minimize the sum of all required relaxations in first phase only  
1: Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations  
2: Minimize the number of constraints and bounds requiring relaxation in first phase only  
3: Minimize the sum of squares of required relaxations in first phase only  
4: Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations | 0       |
| Subsolver.Cplex.MIPEmphasis  | Sets the MIP emphasis  
0: Balanced  
1: Feasibility  
2: Optimality  
3: Best bound  
4: Hidden feasible | 1       |
| Subsolver.Cplex.MemoryEmphasis | Try to conserve memory when possible  
Range: \{0, ..., 1\} | 0       |
| Subsolver.Cplex.NodeFile     | Where to store the node file  
0: No file  
1: Compressed in memory  
2: On disk  
3: Compressed on disk | 1       |
Range: \{0, ..., 1\} | 1       |
| Subsolver.Cplex.OptimalityTarget | Specifies how CPLEX treats nonconvex quadratics  
0: Automatic  
1: Searches for a globally optimal solution to a convex model  
2: Searches for a solution that satisfies first-order optimality conditions, but is not necessarily globally optimal  
3: Searches for a globally optimal solution to a nonconvex model | 0       |
| Subsolver.Cplex.ParallelMode | Controls how much time and memory should be used when filling the solution pool  
-1: Opportunistic  
0: Automatic  
1: Deterministic | 0       |
| Subsolver.Cplex.Probe        | Sets the MIP probing level  
-1: No probing  
0: Automatic  
1: Moderate  
2: Aggressive  
3: Very aggressive | 0       |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsolver.Cplex.SolutionPoolIntensity</td>
<td>Controls how much time and memory should be used when filling the solution pool 0: Automatic 1: Mild 2: Moderate 3: Aggressive 4: Very aggressive</td>
<td>0</td>
</tr>
<tr>
<td>Subsolver.Cplex.SolutionPoolReplace</td>
<td>How to replace solutions in the solution pool when full 0: Replace oldest 1: Replace worst 2: Find diverse</td>
<td>0</td>
</tr>
<tr>
<td>Subsolver.Gurobi.MIPFocus</td>
<td>MIP focus 0: Automatic 1: Feasibility 2: Optimality 3: Best bound</td>
<td>0</td>
</tr>
<tr>
<td>Subsolver.Gurobi.NumericFocus</td>
<td>MIP focus 0: Automatic 1: Mild 2: Moderate 3: Aggressive</td>
<td>1</td>
</tr>
<tr>
<td>Subsolver.Gurobi.PoolSearchMode</td>
<td>Finds extra solutions 0: No extra effort 1: Try to find solutions 2: Find n best solutions</td>
<td>0</td>
</tr>
<tr>
<td>Subsolver.Gurobi.PoolSolutions</td>
<td>Determines how many MIP solutions are stored Range: {1, ..., 2000000000}</td>
<td>10</td>
</tr>
<tr>
<td>Subsolver.Gurobi.ScaleFlag</td>
<td>Controls model scaling -1: Automatic 0: Off 1: Mild 2: Moderate 3: Aggressive</td>
<td>-1</td>
</tr>
<tr>
<td>Subsolver.Ipopt.MaxIterations</td>
<td>Maximum number of iterations Range: {0, ..., ∞}</td>
<td>1000</td>
</tr>
<tr>
<td>Subsolver.Rootsearch.MaxIterations</td>
<td>Maximal root search iterations Range: {0, ..., ∞}</td>
<td>100</td>
</tr>
<tr>
<td>Subsolver.Rootsearch.Method</td>
<td>Root search method to use 0: TOMS748 1: Bisection</td>
<td>0</td>
</tr>
<tr>
<td>Subsolver.Cplex.SolutionPoolGap</td>
<td>Sets the relative gap filter on objective values in the solution pool Range: [0, 1e+75]</td>
<td>1e+75</td>
</tr>
<tr>
<td>Subsolver.Cplex.WorkMemory</td>
<td>Memory limit for when to start swapping to disk Range: [0, 1e+75]</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.38.3.3 Dual strategy

These settings control the various functionality of the dual strategy in SHOT, i.e., the polyhedral outer approximation utilizing the ESH or ECP algorithms.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| Dual.CutStrategy                | Dual cut strategy  
0: ESH  
1: ECP | 0                                 |
| Dual.ESH.InteriorPoint.CuttingPlane.IterationLimit | Iteration limit for minimax cutting plane solver  
Range: \{1, ..., \infty\} | 100 |
| Dual.ESH.InteriorPoint.CuttingPlane.IterationLimitSubsolver | Iteration limit for minimization  
subsolver  
Range: \{0, ..., \infty\} | 100 |
| Dual.ESH.InteriorPoint.UsePrimalSolution | Utilize primal solution as interior point  
0: No  
1: Add as new  
2: Replace old  
3: Use average | 1 |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual.HyperplaneCuts.MaxPerIteration</td>
<td>Maximal number of hyperplanes to add per iteration</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.ObjectiveRootSearch</td>
<td>When to use the objective root search</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Always</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: IfConvex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Never</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.InfeasibilityRepair.IterationLimit</td>
<td>Max number of infeasible problems repaired without primal objective value</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>improvement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.NumberOfThreads</td>
<td>Number of threads to use in MIP solver: 0: Automatic</td>
<td>GAMS threads</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, 999}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.Presolve.Frequency</td>
<td>When to call the MIP presolve</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Never</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Once</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Always</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.SolutionLimit.ForceOptimalIteration</td>
<td>Iterations without dual bound updates for forcing optimal MIP solution</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.SolutionLimit.IncreasedIterations</td>
<td>Max number of iterations between MIP solution limit increases</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.SolutionLimit.Initial</td>
<td>Initial MIP solution limit</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: ${1, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.SolutionPool.Capacity</td>
<td>The maximum number of solutions in the solution pool</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.Solver</td>
<td>Which MIP solver to use</td>
<td>Cplex, if licensed, otherwise Cbc</td>
</tr>
<tr>
<td></td>
<td>0: Cplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Gurobi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Cbc</td>
<td></td>
</tr>
<tr>
<td>Dual.ReductionCut.MaxIterations</td>
<td>Max number of primal cut reduction without primal improvement</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.Relaxation.Frequency</td>
<td>The frequency to solve an LP problem: 0: Disable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.Relaxation.IterationLimit</td>
<td>The max number of relaxed LP problems to solve initially</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.Relaxation.MaxLazyConstraints</td>
<td>Max number of lazy constraints to add in relaxed solutions in single-tree</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>strategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: ${0, \ldots, \infty}$</td>
<td></td>
</tr>
<tr>
<td>Dual.TreeStrategy</td>
<td>The main strategy to use</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Multi-tree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Single-tree</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.CuttingPlane.ConstraintSelectionFactor</td>
<td>The fraction of violated constraints to generate cutting planes for</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.CuttingPlane.TerminationToleranceAbs</td>
<td>Absolute termination tolerance between LP and linesearch objective</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.CuttingPlane.TerminationToleranceRel</td>
<td>Relative termination tolerance between LP and linesearch objective</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.CuttingPlane.TimeLimit</td>
<td>Time limit for minimax solver</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.MinimaxObjectiveLowerBound</td>
<td>Lower bound for minimax objective variable</td>
<td>-1e+12</td>
</tr>
<tr>
<td></td>
<td>Range: (-∞, 0]</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.InteriorPoint.MinimaxObjectiveUpperBound</td>
<td>Upper bound for minimax objective variable</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Dual.E.SH.Rootsearch.ConstraintTolerance</td>
<td>Constraint tolerance for when not to add individual hyperplanes / cutting planes</td>
<td>1e-08</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.ConstraintSelectionFactor</td>
<td>The fraction of violated constraints to generate supporting hyperplanes / cutting planes</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.MaxConstraintFactor</td>
<td>Rootsearch performed on constraints with values larger than this factor times the maximum value</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-06, 1]</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.CutOff.InitialValue</td>
<td>Initial cutoff value to use</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.CutOff.Tolerance</td>
<td>An extra tolerance for the objective cutoff value (to prevent infeasible subproblems)</td>
<td>1e-05</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.InfeasibilityRepair.TimeLimit</td>
<td>Time limit when repairing infeasible problem</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.NodeLimit</td>
<td>Node limit to use for MIP solver in single-tree strategy</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.OptimalityTolerance</td>
<td>The reduced-cost tolerance for optimality in the MIP solver</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-09, 0.01]</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.SolutionLimit.ForceOptimal.Time</td>
<td>Time (s) without dual bound updates for forcing optimal MIP solution</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Dual.MIP.SolutionLimit.UpdateTolerance</td>
<td>The constraint tolerance for when to update MIP solution limit</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>Dual.ReductionCut.ReductionFactor</td>
<td>The factor used to reduce the cutoff value</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>Dual.Relaxation.TerminationTolerance</td>
<td>Time limit (s) when solving LP problems initially</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Dual.Relaxation.TimeLimit</td>
<td>Time limit (s) when solving LP problems initially</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, \infty])</td>
<td></td>
</tr>
<tr>
<td>Dual.ESH.InteriorPoint.CuttingPlane.Reuse</td>
<td>Reuse valid cutting planes in main dual model</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.ESH.Rootsearch.UniqueConstraints</td>
<td>Allow only one hyperplane per constraint per iteration</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.ESH.Rootsearch.UseMaxFunction</td>
<td>Perform rootsearch on max function, otherwise on individual constraints</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.Delay</td>
<td>Add hyperplane cuts to model only after optimal MIP solution</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.SaveHyperplanePoints</td>
<td>Whether to save the points in the generated hyperplanes list</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.HyperplaneCuts.UseIntegerCuts</td>
<td>Add integer cuts for infeasible integer-combinations for binary problems</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.CutOff.UseInitialValue</td>
<td>Use the initial cutoff value</td>
<td>1, if <code>cutoff</code> is set</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.InfeasibilityRepair.IntegerCuts</td>
<td>Allow feasibility repair of integer cuts</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.InfeasibilityRepair.Use</td>
<td>Enable the infeasibility repair strategy for nonconvex problems</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.Presolve.RemoveRedundantConstraints</td>
<td>Remove redundant constraints (as determined by presolve)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.Presolve.UpdateObtainedBounds</td>
<td>Update bounds (from presolve) to the MIP model</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Dual.MIP.UpdateObjectiveBounds</td>
<td>Update nonlinear objective variable bounds to primal/dual bounds</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### 5.38.3.4 Optimization model

These settings control various aspects of SHOT’s representation for and handling of the provided optimization model.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual.ReductionCut.Use</td>
<td>Enable the dual reduction cut strategy for nonconvex problems</td>
<td>1</td>
</tr>
<tr>
<td>Dual.Relaxation.Use</td>
<td>Initially solve continuous dual relaxations</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model.BoundTightening.FeasibilityBased.MaxIterations</td>
<td>Maximal number of bound tightening iterations</td>
<td>5</td>
</tr>
<tr>
<td>Model.BoundTightening.InitialPOA.CutStrategy</td>
<td>Dual cut strategy</td>
<td>1: ECP</td>
</tr>
<tr>
<td>Model.BoundTightening.InitialPOA.IterationLimit</td>
<td>Iteration limit for POA</td>
<td>50</td>
</tr>
<tr>
<td>Model.BoundTightening.InitialPOA.StagnationIterationLimit</td>
<td>Limit for iterations without significant progress</td>
<td>5</td>
</tr>
<tr>
<td>Model.Reformulation.Bilinear.IntegerFormulation</td>
<td>Reformulate integer bilinear terms</td>
<td>1</td>
</tr>
<tr>
<td>Model.Reformulation.Bilinear.IntegerFormulation.MaxDomain</td>
<td>Do not reformulate integer variables in bilinear terms which can assume more than this number of discrete values</td>
<td>100</td>
</tr>
<tr>
<td>Model.Reformulation.Constraint.PartitionNonlinearTerms</td>
<td>When to partition nonlinear sums in constraints</td>
<td>1</td>
</tr>
<tr>
<td>Model.Reformulation.Constraint.PartitionQuadraticTerms</td>
<td>When to partition quadratic sums in constraints</td>
<td>1</td>
</tr>
<tr>
<td>Model.Reformulation.Monomials.Formulation</td>
<td>How to reformulate binary monomials</td>
<td>1</td>
</tr>
<tr>
<td>Model.Reformulation.ObjectiveFunction.PartitionNonlinearTerms</td>
<td>When to partition nonlinear sums in objective function</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Model.Reformulation.ObjectiveFunction.PartitionQuadraticTerms</td>
<td>When to partition quadratic sums in objective function</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Always</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: If result is convex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Never</td>
<td></td>
</tr>
<tr>
<td>Model.Reformulation.Quadratics.EigenValueDecomposition.Formulation</td>
<td>Which formulation to use in eigenvalue decomposition</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: Term coefficient is included in reformulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Term coefficient remains</td>
<td></td>
</tr>
<tr>
<td>Model.Reformulation.Quadratics.ExtractStrategy</td>
<td>How to extract quadratic terms from nonlinear expressions</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Do not extract</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Extract to same objective or constraint</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Extract to quadratic equality constraint if nonconvex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Extract to quadratic equality constraint even if convex</td>
<td></td>
</tr>
<tr>
<td>Model.Reformulation.Quadratics.Strategy</td>
<td>How to treat quadratic functions</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0: All nonlinear</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Use quadratic objective</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Use convex quadratic objective and constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: Use nonconvex quadratic objective and constraints</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.FeasibilityBased.TimeLimit</td>
<td>Time limit for bound tightening</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞)</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.ConstraintTolerance</td>
<td>Constraint termination tolerance</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.ObjectiveConstraintTolerance</td>
<td>Objective constraint termination tolerance</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.ObjectiveGapAbsolute</td>
<td>Absolute objective gap termination level</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.ObjectiveGapRelative</td>
<td>Relative objective gap termination level</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.StagnationConstraintTolerance</td>
<td>Tolerance factor for when no progress is made</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model_BOUND_Tightening.InitialPOA.TimeLimit</td>
<td>Time limit for initial PO</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model.Convexity.Quadratics.EigenValueTolerance</td>
<td>Convexity tolerance for the eigenvalues of the Hessian matrix for quadratic terms</td>
<td>1e-05</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>Model.Reformulation.Quadratics.EigenValueDecomposition.Tolerance</td>
<td>Variables with eigenvalues smaller than this value will be ignored</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [0, ∞]</td>
<td></td>
</tr>
<tr>
<td>Model.Variables.Continuous.MaximumUpperBound</td>
<td>Maximum upper bound for continuous variables</td>
<td>1e+50</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model.Variables.Continuous.MinimumLowerBound</td>
<td>Minimum lower bound for continuous variables</td>
<td>-1e+50</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Model.Variables.Integer.MaximumUpperBound</td>
<td>Maximum upper bound for integer variables</td>
<td>2e+09</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
</tbody>
</table>
### 5.38.3.5 Modeling system

These settings control functionality used in the interfaces to different modeling environments.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelingSystem.GAMS.QExtractAlg</td>
<td>Extraction algorithm for quadratic equations in GAMS interface</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 5.38.3.6 Primal heuristics

These settings control the primal heuristics used in SHOT.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primal.FixedInteger.CallStrategy</td>
<td>When should the fixed strategy be used</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0: Use each iteration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Based on iteration or time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Based on iteration or time, and for all feasible MIP solutions</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Frequency.Iteration</td>
<td>Max number of iterations between calls</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \ldots, \infty}</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.IterationLimit</td>
<td>Max number of iterations per call</td>
<td>10000000</td>
</tr>
<tr>
<td></td>
<td>Range: {0, \ldots, \infty}</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Solver</td>
<td>NLP solver to use</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0: Ipopt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: GAMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: SHOT</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Source</td>
<td>Source of fixed MIP solution point</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0: All</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: First</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: All feasible</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3: First and all feasible</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4: With smallest constraint deviation</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.SourceProblem</td>
<td>Which problem formulation to use for NLP problem</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: Original problem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Reformulated problem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Both</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.DualPointGap.Relative</td>
<td>If the objective gap between the MIP point and dual solution is less than</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>this the fixed strategy is activated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Frequency.Time</td>
<td>Max duration (s) between calls</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.TimeUnitLimit</td>
<td>Time limit (s) per NLP problem</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [0, \infty]</td>
<td></td>
</tr>
<tr>
<td>Primal.Tolerance.Integer</td>
<td>Integer tolerance for accepting primal solutions</td>
<td>1e-05</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Primal.Tolerance.LinearConstraint</td>
<td>Linear constraint tolerance for accepting primal solutions</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Primal.Tolerance.NonlinearConstraint</td>
<td>Nonlinear constraint tolerance for accepting primal solutions</td>
<td>1e-05</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.CreateInfeasibilityCut</td>
<td>Create a cut from an infeasible solution point</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Frequency.Dynamic</td>
<td>Dynamically update the call frequency based on success</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.OnlyUniqueIntegerCombinations</td>
<td>Whether to resolve with the same integer combination, e.g. for nonconvex problems with different continuous variable starting points</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
<tr>
<td>Primal.FixedInteger.Use</td>
<td>Use the fixed integer primal strategy</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: boolean</td>
<td></td>
</tr>
</tbody>
</table>
### 5.38.3.7 Termination

These settings control when SHOT will terminate the solution process.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Termination.DualStagnation.IterationLimit</td>
<td>Max number of iterations without significant dual objective value improvement</td>
<td>∞</td>
</tr>
<tr>
<td>Termination.IterationLimit</td>
<td>Iteration limit for main strategy</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>Termination.PrimalStagnation.IterationLimit</td>
<td>Max number of iterations without significant primal objective value improvement</td>
<td>50</td>
</tr>
<tr>
<td>Termination.ConstraintTolerance</td>
<td>Termination tolerance for nonlinear constraints</td>
<td>1e-08</td>
</tr>
<tr>
<td>Termination.DualStagnation.ConstraintTolerance</td>
<td>Min absolute difference between max nonlinear constraint errors in subsequent iterations for termination</td>
<td>1e-06</td>
</tr>
<tr>
<td>Termination.ObjectiveConstraintTolerance</td>
<td>Termination tolerance for the nonlinear objective constraint</td>
<td>1e-08</td>
</tr>
<tr>
<td>Termination.ObjectiveGap.Absolute</td>
<td>Absolute gap termination tolerance for objective function</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>Termination.ObjectiveGap.Relative</td>
<td>Relative gap termination tolerance for objective function</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>Termination.TimeLimit</td>
<td>Time limit (s) for solver</td>
<td>GAMS reslim</td>
</tr>
</tbody>
</table>

### 5.38.3.8 Strategy

Overall strategy parameters used in SHOT.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy.UseRecommendedSettings</td>
<td>Modifies some settings to their recommended values based on the strategy</td>
<td>1</td>
</tr>
</tbody>
</table>
5.39 SNOPT

Author
Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA
Walter Murray, Michael A. Saunders; Department of EESOR, Stanford University, Stanford, CA
Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.39.1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

\[ \text{NP : minimize or maximize } f_0(x) \]

subject to

\[ f(x) \sim b_1 \]

\[ A_L x \sim b_2 \]

\[ l \leq x \leq u, \]  \hspace{1cm} (1)

where \( x \in \mathbb{R}^n \), \( f_0(x) \) is a linear or nonlinear smooth objective function, \( l \) and \( u \) are constant lower and upper bounds, \( f(x) \) is a set of nonlinear constraint functions, \( A_L \) is a sparse matrix, \( \sim \) is a vector of relational operators (\( \leq, \geq \) or \( = \)), and \( b_1 \) and \( b_2 \) are right-hand side constants. \( f(x) \sim b_1 \)s are the nonlinear constraints of the model and \( A_L x \sim b_2 \) form the linear constraints.

The gradients of \( f_0 \) and \( f_i \) are automatically provided by GAMS, using its automatic differentiation engine.

The bounds may include special values \(-\infty \) or \(+\infty \) to indicate \( l_j = -\infty \) or \( u_j = +\infty \) for appropriate \( j \). Free variables have both bounds infinite and fixed variables have \( l_j = u_j \).

SNOPT uses a sequential quadratic programming (SQP) algorithm that obtains search directions from a sequence of quadratic programming subproblems. Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to a linearization of the constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point.

SNOPT is most efficient if only some of the variables enter nonlinearly, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. SNOPT requires relatively few evaluations of the problem functions.

5.39.1.1 Problem Types

If the nonlinear functions are absent, the problem is a linear program (LP) and SNOPT applies the primal simplex method [41] . Sparse basis factors are maintained by LUSOL [82] as in MINOS [141] .

If only the objective is nonlinear, the problem is linearly constrained (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [85] using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [81] [84] .

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [139] [140] . It is suitable for nonlinear problems with thousands of constraints and variables, and is most efficient if only some of the variables enter nonlinearly, or there are relatively few degrees of freedom at a solution (i.e., many constraints are active). However, unlike previous versions of SNOPT, there is no limit on the number of degrees of freedom.
5.39 SNOPT

5.39.1.2 Selecting the SNOPT Solver

If SNOPT is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select SNOPT:

```gams
option nlp=SNOPT;
```

The option statement should appear before the `solve` statement. To be complete, we mention that the solver can be also specified on the command line, as in:

```
> gams camcge nlp=snopt
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

If the model contains non-smooth functions like \( \text{abs} \ (x) \), or \( \text{max}(x, y) \) you can try to get it solved by SNOPT using

```gams
option dnlp=SNOPT;
```

These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they appear away from an optimum.

5.39.2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described by by Gill, Murray and Saunders [86].

5.39.2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```gams
obj.. z =e= sum(i, sqr(r(i)));
model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize \( z \) subject to \( z = \sum_i r_i^2 \) and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize the nonlinear expression \( \sum_i r_i^2 \) directly. This can be achieved by a simple reformulation: \( z \) can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable \( z \) is a free continuous variable (no bounds are defined on \( z \)),
  ```
• $z$ appears linearly in the objective function,
• the objective function is formulated as an equality constraint,
• $z$ is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model chem.gms from the model library solves in 16 iterations. When we add the bound

\[ \text{energy.lo} = 0; \]

on the objective variable energy and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```gams
variables x,y,z;
equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
model m /all/;
option nlp=snopt;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

\[ \min f(x) = (1 + x)^2. \]

### 5.39.2.2 Constraints and slack variables

Problem (1) contains $n$ variables in $x$. Let $m$ be the number of components of $f(x)$ and $A_L x$ combined. The upper and lower bounds on those terms define the general constraints of the problem. SNOPT converts the general constraints to equalities by introducing a set of slack variables $s = (s_1, s_2, ..., s_m)^T$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. Problem (1) can be written in the equivalent form

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u.
\end{align*}
\]

where a maximization problem is cast into a minimization by multiplying the objective function by $-1$.

The linear and nonlinear general constraints become equalities of the form $f(x) - s_N = 0$ and $A_L x - s_L = 0$, where $s_L$ and $s_N$ are known as the *linear* and *nonlinear* slacks.
5.39.2.3 Major iterations

The basic structure of SNOPT’s solution algorithm involves major and minor iterations. The major iterations generate a sequence of iterates \( \{x_k\} \) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate \( \{x_k\} \) a QP subproblem is used to generate a search direction towards the next iterate \( \{x_{k+1}\} \). The constraints of the subproblem are formed from the linear constraints \( A_L x - s_L = 0 \) and the nonlinear constraint linearization

\[
f(x_k) + f'(x_k)(x - x_k) - s_N = 0,
\]

where \( f'(x_k) \) denotes the Jacobian: a matrix whose rows are the first derivatives of \( f(x) \) evaluated at \( x_k \). The QP constraints therefore comprise the \( m \) linear constraints

\[
f'(x_k)x - s_N = -f(x_k) + f'(x_k)x_k,
\]

\[
A_L x - s_L = 0,
\]

where \( x \) and \( s \) are bounded by \( l \) and \( u \) as before. If the \( m \times n \) matrix \( A \) and \( m \)-vector \( b \) are defined as

\[
A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},
\]

then the QP subproblem can be written as

\[
\begin{align*}
\text{QP}_k & : \quad \min_{x,s} \quad q(x,x_k) = g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^TH_k(x - x_k) \\
\text{subject to} & \quad Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u,
\end{align*}
\]

where \( q(x,x_k) \) is a quadratic approximation to a modified Lagrangian function \([85]\). The matrix \( H_k \) is a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration. If some of the variables enter the Lagrangian linearly the Hessian will have some zero rows and columns. If the nonlinear variables appear first, then only the leading \( n_1 \) rows and columns of the Hessian need be approximated, where \( n_1 \) is the number of nonlinear variables.

5.39.2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure. Here, the iterations of the QP solver SQOPT [87] form the minor iterations of the SQP method.

SQOPT uses a reduced-Hessian active-set method implemented as a reduced-gradient method similar to that in MINOS [139].

At each minor iteration, the constraints \( Ax - s = b \) are partitioned into the form

\[
Bx_B + Sx_S + Nx_N = b,
\]

where the basis matrix \( B \) is square and nonsingular and the matrices \( S, N \) are the remaining columns of \( (A - I) \). The vectors \( x_B, x_S, x_N \) are the associated basic, superbasic, and nonbasic components of the variables \( (x,s) \).

The term active-set method arises because the nonbasic variables \( x_N \) are temporarily frozen at their upper or lower bounds, and their bounds are considered to be active. Since the general constraints are satisfied also, the set of active constraints takes the form

\[
\begin{pmatrix} B & S & N \end{pmatrix} \begin{pmatrix} x_B \\ x_S \\ x_N \end{pmatrix} = \begin{pmatrix} b \end{pmatrix}.
\]
where $x_N$ represents the current values of the nonbasic variables. (In practice, nonbasic variables are sometimes frozen at values strictly between their bounds.) The reduced-gradient method chooses to move the superbasic variables in a direction that will improve the objective function. The basic variables ”tag along” to keep $Ax - s = b$ satisfied, and the nonbasic variables remain unaltered until one of them is chosen to become superbasic.

At a nonoptimal feasible point $(x, s)$ we seek a search direction $p$ such that $(x, s) + p$ remains on the set of active constraints yet improves the QP objective. If the new point is to be feasible, we must have $Bp_B + Sp_S + Np_N = 0$ and $p_N = 0$. Once $p_S$ is specified, $p_B$ is uniquely determined from the system $Bp_B = -Sp_S$. It follows that the superbasic variables may be regarded as independent variables that are free to move in any desired direction. The number of superbasic variables ($n_S$ say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, $n_S$ is a measure of how nonlinear the problem is. In particular, $n_S$ need not be more than one for linear problems.

5.39.2.5 The reduced Hessian and reduced gradient

The dependence of $p$ on $p_S$ may be expressed compactly as $p = Zp_S$, where $Z$ is a matrix that spans the null space of the active constraints:

$$
Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}
$$

where $P$ permutes the columns of $(A - I)$ into the order $(B S N)$. Minimizing $q(x, x_k)$ with respect to $p_S$ now involves a quadratic function of $p_S$:

$$
g^T Zp_S + \frac{1}{2} p^T_S Z^T H Z p_S,
$$

where $g$ and $H$ are expanded forms of $g_k$ and $H_k$ defined for all variables $(x, s)$. This is a quadratic with Hessian $Z^T H Z$ (the reduced Hessian) and constant vector $Z^T g$ (the reduced gradient). If the reduced Hessian is nonsingular, $p_S$ is computed from the system

$$
Z^T H Z p_S = -Z^T g.
$$

The matrix $Z$ is used only as an operator, i.e., it is not stored explicitly. Products of the form $Zv$ or $Z^T g$ are obtained by solving with $B$ or $B^T$. The package LUSOL [82] is used to maintain sparse $LU$ factors of $B$ as the $BSN$ partition changes. From the definition of $Z$, we see that the reduced gradient can be computed from

$$
B^T \pi = g_B, \quad Z^T g = g_S - S^T \pi,
$$

where $\pi$ is an estimate of the dual variables associated with the $m$ equality constraints $Ax - s = b$, and $g_B$ is the basic part of $g$.

By analogy with the elements of $Z^T g$, we define a vector of reduced gradients (or reduced costs) for all variables in $(x, s)$:

$$
d = g - \begin{pmatrix} A^T \\ -I \end{pmatrix} \pi, \text{ so that } d_S = Z^T g.
$$

At a feasible point, the reduced gradients for the slacks $s$ are the dual variables $\pi$.

The optimality conditions for subproblem QP$_k$ (2) may be written in terms of $d$. The current point is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for all superbasic variables ($d_S = 0$). In practice, SNOPT requests an approximate QP solution $(\tilde{s}_k, \tilde{\pi}_k)$ with slightly relaxed conditions on $d_j$.

If $d_S = 0$, no improvement can be made with the current $BSN$ partition, and a nonbasic variable with non-optimal reduced gradient is selected to be added to $S$. The iteration is then repeated with $n_S$ increased by one. At all stages, if the step $(x, s) + p$ would cause a basic or superbasic variable to violate
one of its bounds, a shorter step \((x, s) + \alpha p\) is taken, one of the variables is made nonbasic, and \(n_S\) is decreased by one.

The process of computing and testing reduced gradients \(d_N\) is known as pricing (a term introduced in the context of the simplex method for linear programming). Pricing the \(j\)th variable means computing \(d_j = g_j - a_j^T \pi\), where \(a_j\) is the \(j\)th column of \((A - I)\). If there are significantly more variables than general constraints (i.e., \(n \gg m\)), pricing can be computationally expensive. In this case, a strategy known as partial pricing can be used to compute and test only a subset of \(d_N\).

Solving the reduced Hessian system (5) is sometimes expensive. With the option QPSolver Cholesky, an upper-triangular matrix \(R\) is maintained satisfying \(R^T R = Z^T H Z\). Normally, \(R\) is computed from \(Z^T H Z\) at the start of phase 2 and is then updated as the BSN sets change. For efficiency the dimension of \(R\) should not be excessive (say, \(n_S \leq 1000\)). This is guaranteed if the number of nonlinear variables is "moderate". Other QPSolver options are available for problems with many degrees of freedom.

5.39.2.6 The merit function

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function

\[
M(x, s, \pi) = f(x) - \pi^T (F(x) - s_N) + \frac{1}{2} (F(x) - s_N)^T D (F(x) - s_N),
\]

where \(D\) is a diagonal matrix of penalty parameters. If \((x_k, s_k, \pi_k)\) denotes the current solution estimate and \((\bar{x}_k, \bar{s}_k, \bar{\pi}_k)\) denotes the optimal QP solution, the linesearch determines a step \(\alpha_k\) (\(0 < \alpha_k \leq 1\)) such that the new point

\[
\begin{pmatrix}
  x_{k+1} \\
  s_{k+1} \\
  \pi_{k+1}
\end{pmatrix} = \begin{pmatrix}
  x_k \\
  s_k \\
  \pi_k
\end{pmatrix} + \alpha_k \begin{pmatrix}
  \bar{x}_k - x_k \\
  \bar{s}_k - s_k \\
  \bar{\pi}_k - \pi_k
\end{pmatrix}
\]

(7)

gives a sufficient decrease in the merit function. When necessary, the penalties in \(D\) are increased by the minimum-norm perturbation that ensures descent for \(M\) [84]. As in NPSOL, \(s_N\) is adjusted to minimize the merit function as a function of \(s\) prior to the solution of the QP subproblem. For more details, see [81] [54].

5.39.2.7 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

\[
\text{FLP : } \begin{align*}
\text{minimize} & \quad e^T (v + w) \\
\text{subject to} & \quad \ell \leq (A_L x - v + w) \leq u, \ v \geq 0, \ w \geq 0,
\end{align*}
\]

where \(e\) is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called elastic programming. We also describe it as minimizing the \(\ell_1\) norm of the infeasibilities.)

If the linear constraints are infeasible (\(v \neq 0\) or \(w \neq 0\)), SNOPT terminates without computing the nonlinear functions.
If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP (1) as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables \( \pi \) for the nonlinear constraints become large), SNOPT enters “elastic” mode and solves the problem

\[
\text{NP}(\gamma) : \quad \text{minimize} \quad f_0(x) + \gamma e^T(v + w) \\
\text{subject to} \quad \ell \leq \begin{pmatrix} x \\
 f(x) - v + w \\
 A_L x 
\end{pmatrix} \leq u, \quad v \geq 0, \quad w \geq 0,
\]

where \( \gamma \) is a nonnegative parameter (the elastic weight), and \( f(x) + \gamma e^T(v + w) \) is called a composite objective. If \( \gamma \) is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar \( \ell_1 \) formulation of NP is fundamental to the S \( \ell_1 \)QP algorithm of Fletcher [64]. See also Conn [38].

The initial value of \( \gamma \) is controlled by the optional parameter elastic weight.

### 5.39.3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of “restart” information in case a number of related models are solved in a row. Starting from an optimal point of a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the .L and .M fields of variables and equations.

#### 5.39.3.1 Starting points

To specify a starting point for SNOPT use the .L level values in GAMS. For example, to set all variables \( x_{i,j} := 1 \) use \( x.l(i,j)=1; \). The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points \( (x_i, y_i) \):

\[
\text{Example :} \quad \text{minimize} \quad r \\
\text{subject to} \quad (x_i - a)^2 + (y_i - b)^2 \leq r^2, \quad r \geq 0.
\]

This problem can be modeled in GAMS as follows.

```gams
set i points /p1*p10/;
parameters
    x(i)  x coordinates,
    y(i)  y coordinates;
* fill with random data
    x(i) = uniform(1,10);
    y(i) = uniform(1,10);
```

```gams
```
variables
   a       x coordinate of center of circle
   b       y coordinate of center of circle
   r       radius;

equations
   e(i)    points must be inside circle;

   e(i)..  sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

   r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

   \[ x_{\text{min}} = \min_i x_i, \]
   \[ y_{\text{min}} = \min_i y_i, \]
   \[ x_{\text{max}} = \max_i x_i, \]
   \[ y_{\text{max}} = \max_i y_i, \]

then good estimates are

   \[ a = \frac{(x_{\text{min}} + x_{\text{max}})}{2}, \]
   \[ b = \frac{(y_{\text{min}} + y_{\text{max}})}{2}, \]
   \[ r = \sqrt{(a - x_{\text{min}})^2 + (b - y_{\text{min}})^2}. \]

Thus we include in our model:

parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt((a.l-xmin)^2 + (b.l-ymin)^2);

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds, since GAMS will project variable levels onto their bounds as part of executing a solve statement. For example, when a variable is bounded away from zero by a statement like \( Y.LO = 1; \) and \( Y \) is at its default level of zero, the \texttt{SOLVE} statement will set the level \( Y.L \) to 1.

Note: another way to formulate the model would be to minimize \( r^2 \) instead of \( r \). This allows SNOPT to solve the problem even with the default starting point.
5.39.3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the fawley.gms model from the model library:

\[
\begin{align*}
\text{Model } & \text{exxon /all/;} \\
& \text{...} \\
& \text{Solve exxon maximizing profit using lp;} \\
& \text{Solve exxon maximizing profit using lp;} \\
\end{align*}
\]

we observe the following iteration counts:

\[
\begin{array}{l}
\text{SOLVE SUMMARY} \\
\text{MODEL exxon OBJECTIVE profit} \\
\text{TYPE LP DIRECTION MAXIMIZE} \\
\text{SOLVER SNOPT FROM LINE 278} \\
\text{**** SOLVER STATUS 1 NORMAL COMPLETION} \\
\text{**** MODEL STATUS 1 OPTIMAL} \\
\text{**** OBJECTIVE VALUE 2899.2528} \\
\text{RESOURCE USAGE, LIMIT 0.016 1000.000} \\
\text{ITERATION COUNT, LIMIT 24 10000} \\
\end{array}
\]

\[
\begin{array}{l}
\text{SOLVE SUMMARY} \\
\text{MODEL exxon OBJECTIVE profit} \\
\text{TYPE LP DIRECTION MAXIMIZE} \\
\text{SOLVER SNOPT FROM LINE 279} \\
\text{**** SOLVER STATUS 1 NORMAL COMPLETION} \\
\text{**** MODEL STATUS 1 OPTIMAL} \\
\text{**** OBJECTIVE VALUE 2899.2528} \\
\text{RESOURCE USAGE, LIMIT 0.000 1000.000} \\
\text{ITERATION COUNT, LIMIT 0 10000} \\
\end{array}
\]

The first solve takes 24 iterations, while the second solve needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

- \( \text{X.M} = 0 \): basic
- \( \text{X.M} \neq 0 \): nonbasic if level value is at bound, superbasic otherwise

A marginal value of \( \text{EPS} \) means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the .M values. It is further noted that if too many .M values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the GAMS bratio option.
5.39 SNOPT

5.39.4 GAMS Options

The usual GAMS options (e.g. reslim, sysout) can be used to control GAMS/SNOPT. For more details, see section Controlling a Solver via GAMS Options. We highlight some of the details of this usage below for cases of special interest.

**iterlim**

Sets the minor iteration limit. SNOPT will stop as soon as the number of minor iterations exceeds the iteration limit, in which case the current solution will be reported.

**domlim**

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating the real power $x^y$ for $x < \varepsilon$ ($x^y$ is evaluated as $\exp(y \log x)$). When such an error occurs the count of domain errors is incremented: SNOPT will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and SNOPT continues. For example, in the case of $\sqrt{x}, x < 0$ a zero is passed back for the function value and a large value for the derivative. In many cases SNOPT will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like $x.lo = 0.001$.

**bratio**

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to SNOPT so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead crash a new basis. The bratio determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

**workfactor**

By default, GAMS/SNOPT computes an estimate of the amount of workspace needed by SNOPT, and passes this workspace on to SNOPT for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc. In most cases this is sufficient to solve the model. In some rare cases SNOPT may need more memory, and the user can provide this by specifying a value of workfactor greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to SNOPT for the solve.

5.39.5 SNOPT Options

The performance of GAMS/SNOPT is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-default values for some or all of the options via the SNOPT option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

Note that the option file is not case sensitive. The tables below contain summary information about the SNOPT options, default values, and links to more detailed explanations.

5.39.5.1 Printing
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>major print level</td>
<td>Amount of information printed during optimization (listing file)</td>
<td>1</td>
</tr>
<tr>
<td>minor print level</td>
<td>Amount of information printed during optimization (listing file)</td>
<td>1</td>
</tr>
<tr>
<td>print frequency</td>
<td>Number of iterations between each log line (listing file)</td>
<td>100</td>
</tr>
<tr>
<td>solution</td>
<td>Prints SNOPT solution (listing file)</td>
<td>NO</td>
</tr>
<tr>
<td>summary frequency</td>
<td>Number of iterations between each log line (log file)</td>
<td>100</td>
</tr>
<tr>
<td>suppress parameters</td>
<td>Suppress printing of parameters (listing file)</td>
<td></td>
</tr>
<tr>
<td>system information</td>
<td>Provides additional information on the progress of the iterations (listing file)</td>
<td>NO</td>
</tr>
<tr>
<td>timing level</td>
<td>Amount of timing information (listing file)</td>
<td>3</td>
</tr>
</tbody>
</table>

5.39.5.2 Problem specification

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>feasible point</td>
<td>Ignore objective function and find a feasible point</td>
<td></td>
</tr>
<tr>
<td>infinite bound</td>
<td>Bounds larger than this number are considered Infinity</td>
<td>1.0e20</td>
</tr>
</tbody>
</table>

5.39.5.3 Convergence tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>major feasibility tolerance</td>
<td>Feasibility tolerance for nonlinear constraints</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>major optimality tolerance</td>
<td>Specifies the final accuracy of the dual variables</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>minor feasibility tolerance</td>
<td>Feasibility tolerance for all variables and linear constraints</td>
<td>1.0e-6</td>
</tr>
</tbody>
</table>

5.39.5.4 Derivative checking

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>verify level</td>
<td>Finite-difference checks on the derivatives</td>
<td>-1</td>
</tr>
</tbody>
</table>

5.39.5.5 Scaling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale option</td>
<td>Controls problem scaling</td>
<td>auto</td>
</tr>
<tr>
<td>scale print</td>
<td>Print scaling factors (listing file)</td>
<td></td>
</tr>
<tr>
<td>scale tolerance</td>
<td>Scale tolerance</td>
<td>0.9</td>
</tr>
</tbody>
</table>
5.39.5.6 Other tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash tolerance</td>
<td>Allow crash procedure to ignore small elements in eligible columns</td>
<td>0.1</td>
</tr>
<tr>
<td>linesearch tolerance</td>
<td>Controls accuracy of steplength selected</td>
<td>0.9</td>
</tr>
<tr>
<td>pivot tolerance</td>
<td>Used to keep the basis non-singular</td>
<td>3.67e-11</td>
</tr>
</tbody>
</table>

5.39.5.7 QP subproblems

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash option</td>
<td>Controls the basis crash algorithm</td>
<td>auto: 0 or 3</td>
</tr>
<tr>
<td>elastic weight</td>
<td>Control for elastic mode</td>
<td>1.0e4</td>
</tr>
<tr>
<td>iterations limit</td>
<td>Minor iteration limit</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>partial price</td>
<td>Number of segments in partial pricing strategy</td>
<td>auto</td>
</tr>
<tr>
<td>qpsolver</td>
<td>Controls method used for QP subproblems</td>
<td>Cholesky</td>
</tr>
</tbody>
</table>

5.39.5.8 SQP method

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>central difference interval</td>
<td>Not applicable: GAMS provides analytic derivatives</td>
<td>6.0e-6</td>
</tr>
<tr>
<td>cold start</td>
<td>Ignore advanced basis and use CRASH procedure</td>
<td></td>
</tr>
<tr>
<td>derivative linesearch</td>
<td>Linesearch method (safeguarded cubic interpolation) with use of derivatives</td>
<td></td>
</tr>
<tr>
<td>difference interval</td>
<td>Not applicable: GAMS provides analytic derivatives</td>
<td>1.5e-8</td>
</tr>
<tr>
<td>function precision</td>
<td>Relative accuracy with which the nonlinear functions are evaluated</td>
<td>3.00e-13</td>
</tr>
<tr>
<td>major iterations limit</td>
<td>Max number of major iterations</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>major step limit</td>
<td>Limits the change in x during a linesearch</td>
<td>2.0</td>
</tr>
<tr>
<td>minor iterations limit</td>
<td>Max number of minor iterations between linearizations of nonlinear constraints</td>
<td>500</td>
</tr>
<tr>
<td>new superbasics limit</td>
<td>Limit on new superbasics when a QP subproblem is solved</td>
<td>99</td>
</tr>
<tr>
<td>nonderivative linesearch</td>
<td>Linesearch method (safeguarded quadratic interpolation) without use of derivatives</td>
<td></td>
</tr>
<tr>
<td>penalty parameter</td>
<td>Initial penalty parameter</td>
<td>0</td>
</tr>
<tr>
<td>proximal point method</td>
<td>Controls proximal point method used for solving linear constraints</td>
<td>1</td>
</tr>
<tr>
<td>reduced hessian dimension</td>
<td>Size of Hessian matrix</td>
<td>auto</td>
</tr>
<tr>
<td>superbasics limit</td>
<td>Maximum number of superbasics</td>
<td>1</td>
</tr>
<tr>
<td>unbounded objective value</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e15</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>unbounded step size</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e18</td>
</tr>
<tr>
<td>violation limit</td>
<td>Limit on maximum constraint violation after the line-search</td>
<td>10</td>
</tr>
<tr>
<td>warm start</td>
<td>Use advanced basis provided by GAMS</td>
<td></td>
</tr>
</tbody>
</table>

### 5.39.5.9 Hessian approximation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian frequency</td>
<td>How often the full Hessian is reset to the identity matrix</td>
<td>999999</td>
</tr>
<tr>
<td>hessian full memory</td>
<td>Approximate Hessian is treated as a dense matrix</td>
<td></td>
</tr>
<tr>
<td>hessian limited memory</td>
<td>Limited-memory procedure is used to update a diagonal Hessian approximation</td>
<td></td>
</tr>
<tr>
<td>hessian updates</td>
<td>How often the limited memory Hessian is reset</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.39.5.10 Frequencies

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>check frequency</td>
<td>Controls frequency of linear constraint satisfaction test</td>
<td>60</td>
</tr>
<tr>
<td>expand frequency</td>
<td>Setting for anti-cycling mechanism</td>
<td>10000</td>
</tr>
<tr>
<td>factorization frequency</td>
<td>Number of iterations between basis factorizations</td>
<td>auto: 100 or 50</td>
</tr>
</tbody>
</table>

### 5.39.5.11 LUSOL options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU complete pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU density tolerance</td>
<td>Controls when to move to a dense factorization</td>
<td>0.6</td>
</tr>
<tr>
<td>LU factor tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>auto</td>
</tr>
<tr>
<td>LU partial pivoting</td>
<td>LUSOL pivoting strategy</td>
<td>yes</td>
</tr>
<tr>
<td>LU rook pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU singularity tolerance</td>
<td>Protection against ill-conditioned basis matrices</td>
<td>3.2e-11</td>
</tr>
<tr>
<td>LU update tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>auto</td>
</tr>
</tbody>
</table>

**central difference interval (real):** Not applicable: GAMS provides analytic derivatives ←

Default: 6.0e−6

**check frequency** (integer): Controls frequency of linear constraint satisfaction test ←
Every \( i \)th minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution \( x \) satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form \( Ax - s = b \), where \( s \) is the set of slack variables. To perform the numerical test, the residual vector \( r = b - Ax + s \) is computed. If the largest component of \( r \) is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

**Check frequency** 1 is useful for debugging purposes, but otherwise this option should not be needed.

Range: \( \{1, \ldots, \infty\} \)

Default: 60

**cold start** (no value): Ignore advanced basis and use CRASH procedure

Requests that the CRASH procedure be used to choose an initial basis. This option takes precedence over the GAMS *bratio* option.

**crash option** (integer): Controls the basis crash algorithm

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix \( (A - I) \). The Crash option \( i \) determines which rows and columns of \( A \) are eligible initially, and how many times CRASH is called. Columns of \(-I\) are used to pad the basis where necessary.

If \( i \geq 1 \), certain slacks on inequality rows are selected for the basis first. (If \( i \geq 2 \), numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of \( A \), searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For the remaining unassigned rows, slack variables are inserted to complete the basis.

By default, crash option 3 is used for linearly constrained problems and crash option 0 for problems with nonlinear constraints.

Default: auto: 0 or 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial basis will be a slack basis. The initial basis contains only slack variables: ( B = I ).</td>
</tr>
<tr>
<td>1</td>
<td>One phase CRASH. CRASH is called once, looking for a triangular basis in all rows and columns of the matrix ( A ).</td>
</tr>
<tr>
<td>2</td>
<td>Two phase CRASH. CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).</td>
</tr>
<tr>
<td>3</td>
<td>Three phase CRASH. CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat <em>linear equalities</em> and <em>linear inequalities</em> separately. As before, the last call treats nonlinear rows before the first major iteration.</td>
</tr>
</tbody>
</table>
**Crash Tolerance (real):** Allow crash procedure to ignore small elements in eligible columns

The Crash Tolerance $r$ allows the starting procedure CRASH to ignore certain small nonzeros in each column of $A$. If $a_{\text{max}}$ is the largest element in column $j$, other nonzeros $a_{ij}$ in the column are ignored if $|a_{ij}| \leq a_{\text{max}} \times r$. (To be meaningful, $r$ should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of $A$ and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first $m$ columns of $A$ are the matrix shown under LU_factor_tolerance, i.e. a tridiagonal matrix with entries $-1, 2, -1$. To help CRASH choose all $m$ columns for the initial basis, we would specify Crash tolerance $r$ for some value of $r > 0.5$.

Range: $[0, 1]$  
Default: 0.1

**Derivative Linesearch (no value):** Linesearch method (safeguarded cubic interpolation) with use of derivatives

At each major iteration a linesearch is used to improve the merit function. A Derivative linesearch uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step size $\alpha_k$.

**Difference Interval (real):** Not applicable: GAMS provides analytic derivatives

This alters the interval $h_1$ that is used to estimate gradients by forward differences in the following circumstances:

- In the initial ("cheap") phase of verifying the problem derivatives.
- For verifying the problem derivatives.
- For estimating missing derivatives.

In all cases, a derivative with respect to $x_j$ is estimated by perturbing that component of $x$ to the value $x_j + h_1(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(h_1)$ unless the functions are badly scaled.Judicious alteration of $h_1$ may sometimes lead to greater accuracy. This option has limited use in a GAMS environment as GAMS provides analytical gradients.

Default: $1.5e-8$

**Elastic Weight (real):** Control for elastic mode

The elastic_weight $\omega$ determines the initial weight $\gamma$ associated with problem NP($\gamma$).

At any given major iteration $k$, elastic mode is started if the QP subproblem is infeasible or if the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where $g$ is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.

Thereafter, $\gamma$ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP($\gamma$), but not feasible for NP. After the rth such increase, $\gamma = \omega 10^r(1+\|g(x_{k1})\|_2)$, where $x_{k1}$ is the iterate at which $\gamma$ was first needed.

Default: $1.0e4$

**Expand Frequency (integer):** Setting for anti-cycling mechanism
This option is part of the EXPAND anti-cycling procedure [83] designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the minor feasibility tolerance is $\delta$ and the expand frequency is $k$. Over a period of $k$ iterations, the tolerance actually used by SNOPT increases from $0.5\delta$ to $\delta$ (in steps of $0.5\delta/k$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

At least every $k$ iterations, a resetting procedure eliminates any infeasible nonbasic variables. Increasing $k$ helps to reduce the number of these slightly infeasible nonbasic variables. However, it also diminishes the freedom to choose a large pivot element (see pivot tolerance).

Range: \(\{1, \ldots, \infty\}\)

Default: 10000

**factorization frequency (integer):** Number of iterations between basis factorizations

At most $k$ basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default $k$ is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of $k$ updates is reached.

By default, the frequency is set to 100 for linear models and 50 otherwise.

Range: \(\{1, \ldots, \infty\}\)

Default: auto: 100 or 50

**feasible point (no value):** Ignore objective function and find a feasible point

The keyword **feasible point** means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.

Default: turned off.

**function precision (real):** Relative accuracy with which the nonlinear functions are evaluated

The relative function precision $\epsilon_R$ is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant $x$ and if the first 6 significant digits are known to be correct, the appropriate value for $\epsilon_R$ would be $1.0e^{-6}$.

(Ideally the functions $f_0(x)$ or $f_i(x)$ should have magnitude of order 1. If all functions are substantially less than 1 in magnitude, $\epsilon_R$ should be the absolute precision. For example, if $f(x) = 1.23456789e^{-4}$ at some point and if the first 6 significant digits are known to be correct, the appropriate value for $\epsilon_R$ would be $1.0e^{-10}$.)
• The default value of $\epsilon_R$ is appropriate for simple analytic functions.
• In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate Function precision may lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Default: $3.00\times10^{-13}$

**hessian frequency** *(integer)*: How often the full Hessian is reset to the identity matrix

This option sets the frequency $i$ for resetting the full memory Hessian. For example, if hessian full memory is selected and $i$ BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

Hessian Full Memory and Hessian Frequency = 20 have a similar effect to Hessian Limited Memory and Hessian Updates = 20, except that the latter retains the current diagonal during resets.

Default: 999999

**hessian full memory** *(no value)*: Approximate Hessian is treated as a dense matrix

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If Hessian Full Memory is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables $n_1$ is not too large. In this case, the storage requirement is fixed and one can expect Q-superlinear convergence to the solution.

By default, this storage method is chosen when the number of nonlinear variables $n_1 \leq 75$.

**hessian limited memory** *(no value)*: Limited-memory procedure is used to update a diagonal Hessian approximation

This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

Hessian Limited Memory should be used on problems where the number of nonlinear variables $n_1$ is large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation $H_r$ a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after $H_r$ has been reset to their diagonal.)

By default, this storage method is chosen when the number of nonlinear variables $n_1 > 75$.

**hessian updates** *(integer)*: How often the limited memory Hessian is reset

If hessian limited memory is selected and $i$ BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the cost of each QP iteration also increases with the number of updates. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Default: 10
**infinite bound** *(real)*: Bounds larger than this number are considered Infinity ↔

If \( r > 0 \), \( r \) defines the "infinite" bound \( \text{infBnd} \) in the definition of the problem constraints. Any upper bound greater than or equal to \( \text{infBnd} \) will be regarded as plus infinity (and similarly for a lower bound less than or equal to \( -\text{infBnd} \)). If \( r \leq 0 \), the default value is used.

Default: 1.0e20

**iterations limit** *(integer)*: Minor iteration limit ↔

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option, if set, overrides the GAMS *iterlim* specification.

Default: GAMS *iterlim*

**linesearch tolerance** *(real)*: Controls accuracy of steplength selected ↔

This controls the accuracy with which a steplength will be located along the direction of search at each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. The linesearch tolerance \( t \) determines the accuracy to which this target value is approximated.

- Larger values like \( t = 0.9 \) request just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate, as is usually the case for GAMS models, a more accurate search may be appropriate; try \( t = 0.1, 0.01 \) or \( 0.001 \). The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try \( t = 0.99 \). The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.

Range: \([0, 1]\)

Default: 0.9

**LU complete pivoting** *(no value)*: LUSOL pivoting strategy ↔

See LU\_partial\_pivoting.

**LU density tolerance** *(real)*: Controls when to move to a dense factorization ↔

The density tolerance \( r_1 \) is used during LUSOL’s basis factorization \( B = LU \). Columns of \( L \) and rows of \( U \) are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds \( r_1 \), the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

See also LU\_singularity\_tolerance.

Range: \([0, 1]\)

Default: 0.6

**LU factor tolerance** *(real)*: Trade-off between stability and sparsity in basis factorization ↔
LU factor tolerance  $r_1$

LU update tolerance  $r_2$

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $r_1, r_2 \geq 1.0$. The matrix $L$ is a product of matrices of the form

$$\begin{pmatrix} 1 & 0 \\ \mu & 1 \end{pmatrix},$$

where the multipliers $\mu$ satisfy $|\mu| \leq r_i$. Smaller values of $r_i$ favor stability, while larger values favor sparsity.

- For large and relatively dense problems, smaller values of $r_1$ (e.g., $r_1 = 3.0$) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce $r_1$ and/or $r_2$ in order to achieve stability. For example, if the columns of $A$ include a submatrix of the form

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix},$$

both $r_1$ and $r_2$ should be in the range $1.0 \leq r_i < 2.0$.

For linear models, the defaults are $r_1 = 100$ and $r_2 = 10$, while for nonlinear models both tolerances default to 3.99.

See also LU_update_tolerance.

Range: $[1, \infty)$

Default: auto

**LU partial pivoting (no value):** LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-type search for pivots that locally minimizes fill-in subject to a threshold pivoting stability criterion. The rook pivoting and complete pivoting options are more expensive than partial pivoting but are more stable and better at revealing rank, as long as the LU_factor_tolerance is not too large (say $r_1 < 2.0$).

When numerical difficulties are encountered, SNOPT automatically reduces the $LU$ tolerances toward 1.0 and switches (if necessary) to rook or complete pivoting before reverting to the default or specified options at the next refactorization. (With sysout on and system information enabled, relevant messages are output to the listing file.)

Default: yes

**LU rook pivoting (no value):** LUSOL pivoting strategy

See LU_partial_pivoting.

**LU singularity tolerance (real):** Protection against ill-conditioned basis matrices


The singularity tolerance $r_2$ helps guard against ill-conditioned basis matrices. After $B$ is refactorized, the diagonal elements of $U$ are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the $j$th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart.)

See also LU_density_tolerance.

Default: $3.2 \times 10^{-11}$

**LU update tolerance (real):** Trade-off between stability and sparsity in basis factorization

See LU_factor_tolerance for details.

Range: $[1, \infty]$  

Default: auto

**major feasibility tolerance (real):** Feasibility tolerance for nonlinear constraints

This tolerance $\epsilon_r$ specifies how accurately the nonlinear constraints should be satisfied. The default value of $1.0 \times 10^{-6}$ is appropriate when the constraints are expected to have at least that accuracy.

Let $rowerr$ be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$rowerr = \max_i \frac{\text{viol}_i}{\|x\|} \leq \epsilon_r,$$

where $\text{viol}_i$ is the violation of the $i$th nonlinear constraint.

In the GAMS/SNOPT iteration log, $rowerr$ appears as the quantity labeled "Feasible". If some of the problem functions are known to be of low accuracy, a larger Major feasibility tolerance may be appropriate.

Default: $1.0 \times 10^{-6}$

**major iterations limit (integer):** Max number of major iterations

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints. By default it is set to $\max(1000, m)$.

Default: GAMS iterlim

**major optimality tolerance (real):** Specifies the final accuracy of the dual variables

This tolerance $\epsilon_d$ specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution $(x, s, \pi)$ such that

$$\max_j \frac{\text{Comp}_j}{\|\pi\|} \leq \epsilon_d,$$

where $\text{Comp}_j$ is an estimate of the complementarity slackness for variable $j$. The values $\text{Comp}_j$ are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where $g_j$ is the $j$th component of the objective gradient, $a_j$ is the associated column of the constraint matrix $(A - I)$ and $\pi$ is the set of QP dual variables):

$$\text{Comp}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, $\text{maxComp}$ appears as the quantity labeled "Optimal".

Default: $1.0 \times 10^{-6}$
**major print level (integer):** Amount of information printed during optimization (listing file) ←

This controls the amount of output to the GAMS listing file at each major iteration. This output is only visible if the `sysout` option is turned on. **Major print level 1** gives normal output for linear and nonlinear problems, and **Major print level 11** gives additional details of the Jacobian factorization that commences each major iteration. In general, the value specified may be thought of as a binary number of the form

```
Major print level JFDXbs
```

where each letter stands for a digit that is either 0 or 1 as follows:

- **s** single line that gives a summary of each major iteration. (This entry in `JFDXbs` is not strictly binary since the summary line is printed whenever `JFDXbs ≥ 1`).
- **b** BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if `JFDXbs ≥ 10`).
- **X** `x_k`, the nonlinear variables involved in the objective function or the constraints.
- **D** `π_k`, the dual variables for the nonlinear constraints.
- **F** `F(x_k)`, the values of the nonlinear constraint functions.
- **J** `J(x_k)`, the Jacobian.

To obtain output of any items `JFDXbs`, set the corresponding digit to 1, otherwise to 0.

If J=1, the Jacobian will be output column-wise at the start of each major iteration. Column `j` will be preceded by the value of the corresponding variable `x_j` and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if J=1, there is no reason to specify X=1 unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3  1.250000D+01 BS  1  1.00000E+00  4  2.00000E+00
```

which would mean that `x_3` is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

**Major print level 0** suppresses most output, except for error messages.

Default: 1

**major step limit (real):** Limits the change in `x` during a linesearch ←

This parameter `r` limits the change in `x` during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step `α` over the range `0 < α ≤ β`, where `β` is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on `x` if all the constraints are linear. Normally, the first steplength tried is `α_1 = min(1, β)`.
2. In some cases, such as `f(x) = ae^{bx}` or `f(x) = ax^b`, even a moderate change in the components of `x` can lead to floating-point overflow. The parameter `r` is therefore used to define a limit `β = r(1 + ∥x∥)/∥p∥` (where `p` is the search direction), and the first evaluation of `f(x)` is at the potentially smaller steplength `α_1 = min(1, β, β)`.
3. Wherever possible, upper and lower bounds on `x` should be used to prevent evaluation of nonlinear functions at meaningless points. The **Major step limit** provides an additional safeguard. The default value `r = 2.0` should not affect progress on well behaved problems, but setting `r = 0.1` or `0.01` may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for \( r \) may help locate an optimum near the starting point.

Default: 2.0

**minor feasibility tolerance** *(real)*: Feasibility tolerance for all variables and linear constraints

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within this tolerance \( t \). This includes slack variables, so general linear constraints should also be satisfied to within \( t \).

Feasibility with respect to nonlinear constraints is judged by the **major feasibility tolerance**.

- If the bounds and linear constraints cannot be satisfied to within \( t \), the problem is declared **infeasible**. Let \( s_{\text{Inf}} \) be the corresponding sum of infeasibilities. If \( s_{\text{Inf}} \) is quite small, it may be appropriate to raise \( t \) by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if \( f(x) = \sqrt{x_1} + \log x_2 \), it is essential to place lower bounds on both variables. If \( t = 1.0e^{-6} \), the bounds \( x_1 \geq 10^{-5} \) and \( x_2 \geq 10^{-4} \) might be appropriate. (The log singularity is more serious. In general, keep \( x \) as far away from singularities as possible.)
- If the model is scaled (see **scale option**), feasibility is defined in terms of the **scaled** problem.
- In reality, SNOPT uses \( t \) as a feasibility tolerance for satisfying the bounds on \( x \) and \( s \) in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in **elastic mode** thereafter (with only the linearized nonlinear constraints defined to be elastic). See **elastic weight** for details.

Default: 1.0e-6

**minor iterations limit** *(integer)*: Max number of minor iterations between linearizations of nonlinear constraints

Minor iterations limit \( k \). If the number of minor iterations for the optimality phase of the QP subproblem exceeds \( k \), then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality. Note that more than \( k \) minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the linesearch. In the major iteration log, a \( t \) at the end of a line indicates that the corresponding QP was artificially terminated using the limit \( k \). Note that **iterations limit** defines an independent absolute limit on the total number of minor iterations (summed over all QP subproblems).

Default: 500

**minor print level** *(integer)*: Amount of information printed during optimization (listing file)

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. It is only useful if the GAMS **sysout** option is turned on. The value of \( k \) has the following effect:

- 0 No minor iteration output except error messages.
- \( \geq 1 \) A single line of output each minor iteration (controlled by **print frequency**).
• ≥ 10 Basis factorization statistics generated during the periodic refactorization of the basis (see factorization frequency). Statistics for the first factorization of each major iteration are controlled by the major print level.

Default: 1

new superbasics limit (integer): Limit on new superbasics when a QP subproblem is solved

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than new superbasics limit the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality. In the major iteration log, a "T" at the end of a line indicates that the QP was terminated early in this way.

Default: 99

nondervative linesearch (no value): Linesearch method (safeguarded quadratic interpolation) without use of derivatives

A nondervative linesearch can be slightly less robust on difficult problems, and it is recommended that the default derivative linesearch be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nondervative linesearch may give a significant decrease in computation time. In a GAMS environment derivative linesearch (the default) is more appropriate.

partial price (integer): Number of segments in partial pricing strategy

This parameter sets the number of segments \( k \) using in partial pricing and is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (i.e. when a nonbasic variable is selected to become superbasic).

• When \( k = 1 \), all columns of the constraint matrix \((A - I)\) are searched.
• Otherwise, \( A \) and \( I \) are partitioned to give \( k \) roughly equal segments \( A_j, I_j \) \((j = 1 \text{ to } k)\). If the previous pricing search was successful on \( A_j, I_j \), the next search begins on the segments \( A_{j+1}, I_{j+1} \). (All subscripts here are modulo \( k \).)
• If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments \( A_{j+2}, I_{j+2} \), and so on.
• Partial price \( t \) (or \( t/2 \) or \( t/3 \)) may be appropriate for time-stage models having \( t \) time periods.

The default is 10 for linear models and 1 for nonlinear models.

Range: \{1, ..., \( \infty \}\}

Default: auto

penalty parameter (real): Initial penalty parameter

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function. This functions contains penalty parameters, which may be increased to ensure descent.

Default: 0

pivot tolerance (real): Used to keep the basis non-singular
During solution of QP subproblems, the pivot tolerance $r$ is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When $x$ changes to $x + \alpha p$ for some search direction $p$, a ratio test is used to determine which component of $x$ reaches an upper or lower bound first. The corresponding element of $p$ is called the pivot element.
- Elements of $p$ are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance $r$.
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the minor feasibility tolerance (say $t$) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of $t$ should therefore not be specified.
- To a lesser extent, the expand frequency (say $f$) also provides some freedom to maximize the pivot element. Excessively large values of $f$ should therefore not be specified.

Default: $3.67\times 10^{-11}$

**print frequency** *(integer)*: Number of iterations between each log line (listing file)

Synonym: log frequency

When `sysout` is turned on and `minor print level` is positive, a line of the QP iteration log will be printed on the listing file every $k$th minor iteration.

Range: \{1, ..., $\infty$\}

Default: 100

**proximal point method** *(integer)*: Controls proximal point method used for solving linear constraints

Once the linear constraints are satisfied, the proximal point method chooses a linear feasible point that is closest to $x_0$, the initial point for the nonlinear variables. The idea is to both satisfy the linear constraints and stay close to the starting values provided for the nonlinear variables. This option is used to disable the proximal point method or to select the norm used.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disable PPM</td>
</tr>
<tr>
<td>1</td>
<td>one-norm</td>
</tr>
</tbody>
</table>

Minimize the one-norm $\|x - x_0\|_1$. | 2 | two-norm
Minimize the two-norm $\frac{1}{2}\|x - x_0\|_2^2$.

**qpsolver** *(string)*: Controls method used for QP subproblems

This specifies the method used to solve system (5) for the search directions in phase 2 of the QP subproblem.

- The Cholesky QP solver is the most robust, but may require a significant amount of computation if the number of superbasics is large.
The quasi-Newton QP solver does not require the computation of the $R$ at the start of each QP subproblem. It may be appropriate when the number of superbasics is large but relatively few major iterations are needed to reach a solution (e.g., if SNOPT is called with a warm start).

The conjugate-gradient QP solver is appropriate for problems with many degrees of freedom (say, more than 2000 superbasics).

Default: Cholesky

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholesky</td>
<td>full Cholesky factor</td>
</tr>
<tr>
<td></td>
<td><strong>QPSolver Cholesky</strong> holds the full Cholesky factor $R$ of the reduced Hessian $Z^T H Z$. As the minor iterations proceed, the dimension of $R$ changes with the number of superbasic variables. If the number of superbasic variables needs to increase beyond the value of <strong>reduced hessian dimension</strong>, the reduced Hessian cannot be stored and the solver switches to <strong>QPSolver CG</strong>. The Cholesky solver is reactivated if the number of superbasics stabilizes at a value less than the <strong>reduced hessian dimension</strong>.</td>
</tr>
<tr>
<td>QN</td>
<td>quasi-Newton method</td>
</tr>
<tr>
<td></td>
<td><strong>QPSolver QN</strong> solves the QP using a quasi-Newton method similar to that of MINOS. In this case, $R$ is the factor of a quasi-Newton approximate Hessian.</td>
</tr>
<tr>
<td>CG</td>
<td>conjugate-gradient method</td>
</tr>
<tr>
<td></td>
<td><strong>QPSolver CG</strong> uses an active-set method similar to <strong>QPSolver QN</strong>, but uses the conjugate-gradient method to solve all systems involving the reduced Hessian.</td>
</tr>
</tbody>
</table>

**reduced hessian dimension (integer):** Size of Hessian matrix $\leftrightarrow$

Synonym: hessian_dimension

This specifies that an $i \times i$ triangular matrix $R$ is to be available for use by the **QPSolver Cholesky** option (to define the reduced Hessian according to $R^T R = Z^T H Z$). The value of $i$ affects when **QPSolver CG** is activated. The default is computed internally by SNOPT, currently as $\min\{2000, n_1 + 1\}$.

Range: $\{1, ..., \infty\}$

Default: auto

**scale option (integer):** Controls problem scaling $\leftrightarrow$

Three scale options are available. By default, option 2 is used for linear models and option 1 for nonlinear models. See also scale tolerance and scale print.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No scaling</td>
</tr>
<tr>
<td></td>
<td>This is recommended if it is known that $x$ and the constraint matrix and Jacobian never have very large elements (say, larger than 100).</td>
</tr>
<tr>
<td>1</td>
<td>Scale linear variables</td>
</tr>
<tr>
<td></td>
<td>Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [69]). This will sometimes improve the performance of the solution procedures.</td>
</tr>
<tr>
<td>2</td>
<td>Scale linear + nonlinear variables</td>
</tr>
<tr>
<td></td>
<td>All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A - I)$ that are fixed or have positive lower bounds or negative upper bounds. If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. <strong>Scale option 2</strong> should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.</td>
</tr>
</tbody>
</table>
**scale print (no value):** Print scaling factors (listing file) $\leftrightarrow$

*Scale print* causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j-n)$ if $j > n$.

The listing file will only show these values if the *sysout* option is turned on. See also *scale option* and *scale tolerance*.

**scale tolerance (real):** Scale tolerance $\leftrightarrow$

The scale tolerance $t$ affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \quad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than $r$ times its previous value, another scaling pass is performed to adjust the row and column scales. Raising $r$ from 0.9 to 0.99 (say) usually increases the number of scaling passes through $A$. At most 10 passes are made.

See also *scale option* and *scale print*.

Range: [0, 1]

Default: 0.9

**solution (string):** Prints SNOPT solution (listing file) $\leftrightarrow$

This option causes the SNOPT solution file to be printed to the GAMS listing file, provided the *sysout* option is also turned on.

Default: turned off.

Default: NO

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Turn off printing of solution</td>
</tr>
<tr>
<td>YES</td>
<td>Turn on printing of solution</td>
</tr>
</tbody>
</table>

**summary frequency (integer):** Number of iterations between each log line (log file) $\leftrightarrow$

If *minor print level* is positive, a line of the QP iteration log is output every $k$th minor iteration.

Range: $\{1, ..., \infty\}$

Default: 100

**superbasics limit (integer):** Maximum number of superbasics $\leftrightarrow$

This places a limit on the storage allocated for superbasic variables. Ideally, $i$ should be set slightly larger than the number of degrees of freedom expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. The default value of $i$ is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".
Normally, \( i \) need not be greater than \( n_1 + 1 \), where \( n_1 \) is the number of nonlinear variables. For many problems, \( i \) may be considerably smaller than \( n_1 \). This will save storage if \( n_1 \) is very large.

This parameter also sets the reduced hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, based on problem characteristics.

Range: \( \{1, \ldots, \infty\} \)

Default: 1

**suppress parameters** *(no value):* Suppress printing of parameters (listing file)

Normally SNOPT prints the option file as it is being read, and then prints a complete list of the available keywords and their final values. The `suppress parameters` option tells SNOPT not to print the full list. Used in conjunction with the `sysout` option.

**system information** *(string):* Provides additional information on the progress of the iterations (listing file)

The `Yes` option provides additional information on the progress of the iterations, including basis repair details when ill-conditioned bases are encountered and the LU factorization parameters are strengthened.

Default: `NO`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NO</code></td>
<td>Turn off additional printing of information on progress of algorithm</td>
</tr>
<tr>
<td><code>YES</code></td>
<td>Turn on additional printing of information on progress of algorithm</td>
</tr>
</tbody>
</table>

**timing level** *(integer):* Amount of timing information (listing file)

Amount of timing information written to the listing file. Used in conjunction with the `sysout` option.

Range: \( \{0, \ldots, 3\} \)

Default: 3

**unbounded objective value** *(real):* Determines when a problem is called unbounded

This parameter sets the value \( f_{\text{max}} \) intended to detect unboundedness in nonlinear problems. See unbounded step size for the setting of \( \alpha_{\text{max}} \).

During a line search, \( f_0 \) is evaluated at points of the form \( x + \alpha p \), where \( x \) and \( p \) are fixed and \( \alpha \) varies. If \( |f_0| \) exceeds \( f_{\text{max}} \) or \( \alpha \) exceeds \( \alpha_{\text{max}} \), iterations are terminated with the exit message *Problem is unbounded (or badly scaled)*.

Unboundedness in \( x \) is best avoided by placing finite upper and lower bounds on the variables.

Default: \( 1.0 \times 10^{15} \)

**unbounded step size** *(real):* Determines when a problem is called unbounded
This parameter sets the value $\alpha_{\text{max}}$ used in the unboundedness test: see unbounded objective value for details.

Default: $1.0 \times 10^{18}$

**verify level (integer):** Finite-difference checks on the derivatives

This option refers to finite-difference checks on the derivatives computed by the user-provided routines. Derivatives are checked at the first point that satisfies all bounds and linear constraints.

This option has limited use in a GAMS environment.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cheap test</td>
</tr>
<tr>
<td>1</td>
<td>Check individual gradients</td>
</tr>
<tr>
<td>2</td>
<td>Check individual columns of the Jacobian</td>
</tr>
<tr>
<td>3</td>
<td>Combines verify level 1 and 2</td>
</tr>
<tr>
<td>-1</td>
<td>Derivative checking is disabled</td>
</tr>
</tbody>
</table>

**violation limit (integer):** Limit on maximum constraint violation after the linesearch

This parameter $\tau$ is used to define an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate $x_{k+1}$ satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\},$$

where $x_0$ is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the $i$th nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an expanded feasible region whose size depends on the magnitude of $\tau$. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then $\tau$ may be any large positive value.

Default: 10

**warm start (no value):** Use advanced basis provided by GAMS

Use an advanced basis provided by GAMS. This option takes precedence over the GAMS bratio option.

### 5.39.6 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

```
-------------------------------------------------------------------------------
S N O P T  7.7.6      (Jan 2021)
-------------------------------------------------------------------------------
SNMEMA EXIT 100 -- finished successfully
SNMEMA INFO 104 -- memory requirements estimated
```
Nonlinear constraints 0  Linear constraints 5  
Nonlinear variables 11  Linear variables 0  
Jacobian variables 0  Objective variables 11  
Total constraints 5  Total variables 11  
The user has defined 11 out of 11 first derivatives

<table>
<thead>
<tr>
<th>Major</th>
<th>Minors</th>
<th>Step</th>
<th>nObj</th>
<th>Feasible</th>
<th>Optimal</th>
<th>Objective</th>
<th>nS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
<td>4.3E-02</td>
<td>-4.7361869E+01</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3</td>
<td>1.0E+00</td>
<td>2</td>
<td>2.1E-02</td>
<td>-4.7526522E+01</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>3</td>
<td>3.4E-02</td>
<td>-4.7586643E+01</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>4</td>
<td>1.1E-02</td>
<td>-4.7662724E+01</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>5</td>
<td>4.8E-03</td>
<td>-4.7678618E+01</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td>1.0E+00</td>
<td>6</td>
<td>1.3E-02</td>
<td>-4.7701147E+01</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>7</td>
<td>5.2E-03</td>
<td>-4.7704100E+01</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>8</td>
<td>3.5E-04</td>
<td>-4.7705775E+01</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>2</td>
<td>1.0E+00</td>
<td>9</td>
<td>5.1E-03</td>
<td>-4.7706132E+01</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>1.0E+00</td>
<td>10</td>
<td>6.1E-05</td>
<td>-4.7706500E+01</td>
<td>6</td>
</tr>
</tbody>
</table>

Major Minors  Step  nObj  Feasible  Optimal  Objective  nS
10  1  1.0E+00 | 11 | 6.1E-05 | -4.7706513E+01 | 6  |
11  1  1.0E+00 | 12 | 5.0E-05 | -4.7706515E+01 | 6  |
12  1  1.0E+00 | 13 | 1.6E-05 | -4.7706515E+01 | 6  |
13  1  1.0E+00 | 14 | 2.0E-06 | -4.7706515E+01 | 6  |
14  1  1.0E+00 | 15 | (7.1E-07)-4.7706515E+01 | 6  |

SNOPTA EXIT 0  -- finished successfully
SNOPTA INFO 1  -- optimality conditions satisfied

Problem name  mixer
No. of iterations 30  Objective -4.7706514831E+01
No. of major iterations 14  Linear  obj. term 0.0000000000E+00
Nonlinear obj. term -4.7706514831E+01
User function calls (total) 15
No. of superbasics 6  No. of basic nonlinears 4
No. of degenerate steps 0  Percentage 0.00
Max x 11 1.6E+00  Max pi 3 1.5E+01
Max Primal infeas 0 0.0E+00  Max Dual infeas 4 6.1E-05

Solution not printed
Time for MPS input 0.00 seconds
Time for solving problem 0.00 seconds
Time for solution output 0.00 seconds
Time for constraint functions 0.00 seconds
Time for objective function 0.00 seconds

For a nonlinearly constrained problem, the log is somewhat different:

================================================================================================
S N O P T  7.7.6  (Jan 2021)
================================================================================================
SNMEMA EXIT 100  -- finished successfully
SNMEMA INFO 104  -- memory requirements estimated

Nonlinear constraints 23  Linear constraints 16  
Nonlinear variables 35  Linear variables 8  
Jacobian variables 35  Objective variables 0  
Total constraints 39  Total variables 43  
The user has defined 56 out of 56 first derivatives
The SNOPT log shows the following columns:

### Major

- **Major**: The current major iteration number.

### Minors

- **Minors**: The number of minor iterations in the current major iteration.
- **Step**: The step size used in the current major iteration.
- **nCon**: The number of constraints.
- **Feasible**: Whether the solution is feasible.
- **Optimal**: Whether the solution is optimal.
- **MeritFunction**: The merit function value.
- **nS**: The number of superbasics.
- **Penalty**: The penalty parameter.

### Problem Information

- **Problem name**: chenrad
- **No. of iterations**: 171
- **Objective**: 1.0589198561E+03
- **No. of major iterations**: 90
- **Linear obj. term**: 1.0589198561E+03
- **Penalty parameter**: 3.188E+00
- **Nonlinear obj. term**: 0.0000000000E+00
- **User function calls (total)**: 286
- **No. of superbasics**: 2
- **No. of degenerate steps**: 9
- **Max x**: 4.64E+02
- **Max pi**: 11 2.6E+03
- **Max Primal infeas**: 48 4.8E-07
- **Max Dual infeas**: 21 9.9E-04
- **Nonlinear constraint violn**: 4.8E-07

### Time Information

- **Time for MPS input**: 0.00 seconds
- **Time for solving problem**: 0.02 seconds
- **Time for solution output**: 0.00 seconds
- **Time for constraint functions**: 0.01 seconds
- **Time for objective function**: 0.00 seconds

The SNOPT log shows the following columns:
The number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, \textit{Minor} will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section Description of the method)

\textbf{Step}

The step length $\alpha$ taken along the current search direction $p$. The variables $x$ have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

\textbf{nObj}

The number of times the nonlinear objective function has been evaluated. \textit{nObj} is printed as a guide to the amount of work required for the linesearch.

\textbf{nCon}

The number of times SNOPT evaluated the nonlinear constraint functions.

\textbf{MeritFunction}

The value of the augmented Lagrangian merit function \textit{(6)}. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section Description of the method). As the solution is approached, \textit{Merit} will converge to the value of the objective at the solution.

In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.

If the constraints are linear, this item is labeled \textit{Objective}, the value of the objective function. It will decrease monotonically to its optimal value.

\textbf{Feasible}

The value of \textit{rowerr}, the maximum component of the scaled nonlinear constraint residual. The solution is regarded as acceptably feasible if \textit{Feasible} is less than the Major feasibility tolerance.

If the constraints are linear, all iterates are feasible and this entry is not printed.

\textbf{Optimal}

The value of \textit{maxgap}, the maximum complementarity gap. It is an estimate of the degree of nonoptimality of the reduced costs. The solution is considered to be optimal if \textit{Optimal} is less than the major optimality tolerance.

\textbf{nS}

The current number of superbasic variables.

\textbf{Penalty}

The Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

The summary line may include additional code characters that indicate what happened during the course of the iteration.
Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.

During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of violation limit.

The norm-wise change in the variables was limited by the value of the major step limit. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of major step limit.

If SNOPT is not in elastic mode, an "i" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP(γ).

If SNOPT is already in elastic mode, an "i" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

This is the same as "M" except that it was also necessary to modify the update to include an augmented Lagrangian term.

No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the hessian frequency and hessian updates keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

This is the same as a "s" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

The minor iterations were terminated at the minor iterations limit.

The minor iterations were terminated at the new superbasics limit.

The QP subproblem was unbounded.

A weak solution of the QP subproblem was found.

The superbasics limit was reached.

Finally SNOPT prints an exit message. See Section EXIT conditions.
5.39.6.1 EXIT conditions

When the solution procedure terminates, an SNOPTA EXIT and SNOPTA INFO message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

1 -- optimality conditions satisfied

The final point seems to be a solution. This means that $x$ is feasible (it satisfies the constraints to the accuracy requested), the reduced gradient is negligible, the reduced costs are optimal, and $R$ is nonsingular. In all cases, some caution should be exercised. For example, if the objective value is much better than expected, SNOPT may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are minimizing the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the same objective value, particularly if the objective is largely linear.) Conversely, if we are maximizing a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to know something about their problem, and to make use of that knowledge as they themselves know best.

One other caution about the "Optimal solution" message. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. If sysout is on, the listing file will contain several indicators of potential issues. Max Primal infeas indicates the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller minor feasibility tolerance (say 10 times smaller) and perhaps a scale option of 0.

Similarly, Max Dual infeas indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\text{Max Dual infeas/Norm of pi} = 10^{-d},$$

then the objective function would probably change in the $d$th significant digit if optimization could be continued. If $d$ seems too large, consider restarting with smaller values of major feasibility tolerance and minor feasibility tolerance.

Finally, Nonlinear constraint violn shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller major feasibility tolerance.

2 -- feasible point found

Occurs only if feasible point is enabled.

3 -- requested accuracy could not be achieved

If the requested accuracy could not be achieved, a feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within $10^{-2}$ of satisfying the major optimality tolerance. Check that the major optimality tolerance is not too small.
11 -- infeasible linear constraints

12 -- infeasible linear equality constraints

When the constraints are linear, the output messages are based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints \(Ax - s = 0\), there is apparently no point that satisfies the bounds on \(x\) and \(s\). Violations as small as the minor feasibility tolerance are ignored, but at least one component of \(x\) or \(s\) violates a bound by more than the tolerance.

13 -- nonlinear infeasibilities minimized

14 -- linear infeasibilities minimized

15 -- infeasible linear constraints in QP subproblem

16 -- infeasible nonelastic constraints

When nonlinear constraints are present, infeasibility is \textit{much} harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the \textit{elastic weight} parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as \textit{elastic variables}. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

21 -- unbounded objective

22 -- constraint violation limit reached

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Adding a bound on the objective will allow SNOPT to find a solution, and inspection of this solution will show the variables that can become too large due to missing restrictions.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the scale option.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (see \textit{unbounded step size} and \textit{unbounded objective value}), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the \textit{violation limit}. 

31 -- iteration limit reached

An iteration limit was reached. Most often this is cured by increasing the GAMS *iterlim* option. If an SNOPT option file was used, also the *iterations limit* may have been set too small.

Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

32 -- major iteration limit reached

This indicates SNOPT was running out the limit on major iterations. This can be changed using the *major iterations limit*.

33 -- the superbasics limit is too small

The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already as many superbasics as allowed (and no room for any more).

When increasing the superbasics limit, be aware that this also increases the reduced hessian dimension unless both options are set explicitly. This may increase the amount of memory required by SNOPT dramatically. Consider also increasing the amount memory available to SNOPT via the workfactor option, or setting a more moderate value for the reduced hessian dimension option and possibly getting slower convergence.

34 -- time limit reached

A time limit was hit. Increase the GAMS *reslim* option.

41 -- current point cannot be improved

The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the major optimality tolerance will probably make this message go away. Try better scaling, better bounds or a better starting point.

42 -- singular basis

The first factorization attempt found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix $U$ were deemed too small.) The associated variables were replaced by slacks and the modified basis refactored, but singularity persisted. Try better scaling, better bounds or a better starting point.

43 -- cannot satisfy the general constraints

The basic variables $x_B$ have been recomputed, given the present values of the superbasic and nonbasic variables. A step of ”iterative refinement” has also been applied to increase the accuracy of $x_B$, but a row check has revealed that the resulting solution does not satisfy the QP constraints $Ax - s = b$ sufficiently well. Try better scaling, better bounds or a better starting point.

44 -- ill-conditioned null-space basis

During computation of the reduced Hessian $Z^T H Z$, some column(s) of $Z$ continued to contain very large values. Try better scaling, better bounds or a better starting point.
51 -- incorrect objective derivatives

52 -- incorrect constraint derivatives

The derivatives are not deemed to be correct. This message should not occur using a GAMS model without external equations.

53 -- irregular or badly scaled problem functions

The problem function can be badly scaled in case of DNLPs (which SNOPT was not designed for) or because of evaluations at singular points.

61 -- undefined function at the first feasible point

62 -- undefined function at the initial point

SNOPT was unable to proceed because the functions are undefined at the initial point or the first feasible point. Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

63 -- unable to proceed into undefined region

Repeated attempts to move into a region where the functions are not defined resulted in the change in variables being unacceptably small. At the final point, it appears that the only way to decrease the merit function is to move into a region where the problem functions are not defined.

Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

71 -- terminated during function evaluation

72 -- terminated from monitor routine

These messages indicate trouble evaluating the non-linear functions or derivatives. Usually these errors show a “Function evaluation error limit” message.

81 -- work arrays must have at least 500 elements

82 -- not enough character storage

83 -- not enough integer storage

84 -- not enough real storage

Increase the memory allocation by using workfactor > 1.

5.40 SoPlex

SoPlex (Sequential object-oriented simPlex) is an optimization package for solving linear programming problems (LPs) based on an advanced implementation of the primal and dual revised simplex algorithm. SoPlex is developed at the Zuse-Institute Berlin.

For more detailed information, we refer to [74] [127] [88] [89] [75] [206] and the SoPlex web site.
5.40.1 Usage

The following statement can be used inside your GAMS program to specify using SOPLEX

```gams
Option LP = SOPLEX; { or RMIP }
```

The above statement should appear before the Solve statement. If SoPlex was specified as the default solver during GAMS installation, the above statement is not necessary.

5.40.1.1 Specification of SoPlex Options

GAMS/SoPlex supports the GAMS parameters `reslim` and `iterlim`.

To allow to run without an iteration limit (SoPlex default), the GAMS/SoPlex link does not pass on a setting of GAMS option `iterlim` to 2147483647.

Setting the GAMS option `integer1` to a nonzero value enables writing of detailed solution statistics to the log.

Setting the GAMS option `integer3` to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process. `integer3=1` writes an MPS file, `integer3=2` writes an LP files, `integer3=4` writes SoPlex state files (.mps, .bas, .set); sum these values to write several files. The name of the output file is chosen to be the name of the GAMS model file with the extension `.gms` replaced. Setting the GAMS option `integer2` to a nonzero value makes variable and equation names available when writing the LP or MPS files. These options may be useful for debugging purposes.

Options can be specified by a SoPlex options file. A SoPlex options file consists of one option or comment per line. A pound sign (`#`) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (`=`) and any amount of white space (blanks or tabs).

A small example for a soplex.opt file is:

```plaintext
bool:rowboundflips = true
int:algorithm = 0
real:feastol = 1e-5
```

It causes GAMS/SoPlex to use bound flipping also for row representations, use the primal simplex, and use a primal feasibility tolerance of 1e-5.
## 5.40.2 List of SoPlex Options

In the following, we give a detailed list of all SoPlex options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool:acceptcycling</td>
<td>should cycling solutions be accepted during iterative refinement?</td>
<td>0</td>
</tr>
<tr>
<td>bool:computedegensd</td>
<td>should the degeneracy be computed for each basis?</td>
<td>0</td>
</tr>
<tr>
<td>bool:decompositiondualnsimplex</td>
<td>should the decomposition based dual simplex be used to solve the LP?</td>
<td>0</td>
</tr>
<tr>
<td>bool:ensureray</td>
<td>re-optimize the original problem to get a proof (ray) of infeasibility/unboundedness?</td>
<td>0</td>
</tr>
<tr>
<td>bool:explicitviol</td>
<td>Should violations of the original problem be explicitly computed in the decomposition simplex?</td>
<td>0</td>
</tr>
<tr>
<td>bool:forcebasic</td>
<td>try to enforce that the optimal solution is a basic solution</td>
<td>0</td>
</tr>
<tr>
<td>bool:fullperturbation</td>
<td>should perturbation be applied to the entire problem?</td>
<td>0</td>
</tr>
<tr>
<td>bool:lifting</td>
<td>should lifting be used to reduce range of nonzero matrix coefficients?</td>
<td>0</td>
</tr>
<tr>
<td>bool:persistentscaling</td>
<td>should persistent scaling be used?</td>
<td>1</td>
</tr>
<tr>
<td>bool:powerscaling</td>
<td>round scaling factors for iterative refinement to powers of two?</td>
<td>1</td>
</tr>
<tr>
<td>bool:ratfacjump</td>
<td>continue iterative refinement with exact basic solution if not optimal?</td>
<td>0</td>
</tr>
<tr>
<td>bool:rowboundflips</td>
<td>use bound flipping also for row representation?</td>
<td>0</td>
</tr>
<tr>
<td>bool:testdualinf</td>
<td>should dual infeasibility be tested in order to try to return a dual solution even if primal infeasible?</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>bool:usecompdual</td>
<td>should the dual of the complementary problem be used in the decomposition simplex?</td>
<td>0</td>
</tr>
<tr>
<td>int:algorithm</td>
<td>type of algorithm (0 - primal, 1 - dual)</td>
<td>1</td>
</tr>
<tr>
<td>int:decomp_displayfreq</td>
<td>the frequency that the decomposition based simplex status output is displayed.</td>
<td>50</td>
</tr>
<tr>
<td>int:decomp_iterlimit</td>
<td>the number of iterations before the decomposition simplex initialisation solve is terminated</td>
<td>100</td>
</tr>
<tr>
<td>int:decomp_maxaddedrows</td>
<td>maximum number of rows that are added to the reduced problem when using the decomposition based simplex</td>
<td>500</td>
</tr>
<tr>
<td>int:decomp verbosity</td>
<td>the verbosity of decomposition based simplex (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full).</td>
<td>0</td>
</tr>
<tr>
<td>int:displayfreq</td>
<td>display frequency</td>
<td>200</td>
</tr>
<tr>
<td>int:factor_update_max</td>
<td>maximum number of LU updates without fresh factorization (0 - auto)</td>
<td>0</td>
</tr>
<tr>
<td>int:factor_update_type</td>
<td>type of LU update (0 - eta update, 1 - Forrest-Tomlin update)</td>
<td>1</td>
</tr>
<tr>
<td>int:hyperpricing</td>
<td>mode for hyper sparse pricing (0 - off, 1 - auto, 2 - always)</td>
<td>1</td>
</tr>
<tr>
<td>int:iterlimit</td>
<td>iteration limit (-1 - no limit)</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>int:leastsq_maxrounds</td>
<td>maximum number of conjugate gradient iterations in least square scaling</td>
<td>50</td>
</tr>
<tr>
<td>int:pricer</td>
<td>pricing method (0 - auto, 1 - dantzig, 2 - parmult, 3 - devex, 4 - quicksteep, 5 - steep)</td>
<td>0</td>
</tr>
<tr>
<td>int:printbasismetric</td>
<td>print basis metric during the solve (-1 - off, 0 - condition estimate, 1 - trace, 2 - determinant, 3 - condition)</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>int:ratiotester</td>
<td>method for ratio test (0 - textbook, 1 - harris, 2 - fast, 3 - boundflipping)</td>
<td>3</td>
</tr>
<tr>
<td>int:reflimit</td>
<td>refinement limit (-1 - no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>int:representation</td>
<td>type of computational form (0 - auto, 1 - column representation, 2 - row representation)</td>
<td>0</td>
</tr>
<tr>
<td>int:scaler</td>
<td>scaling (0 - off, 1 - uniequilibrium, 2 - bi-equilibrium, 3 - geometric, 4 - iterated geometric, 5 - least squares, 6 - geometric-equilibrium)</td>
<td>2</td>
</tr>
<tr>
<td>int:simplifier</td>
<td>simplifier (0 - off, 1 - auto, 2 - PaPILO, 3 - internal)</td>
<td>3</td>
</tr>
<tr>
<td>int:stallreflimit</td>
<td>stalling refinement limit (-1 - no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>int:starter</td>
<td>crash basis generated when starting from scratch (0 - none, 1 - weight, 2 - sum, 3 - vector)</td>
<td>0</td>
</tr>
<tr>
<td>int:stattimer</td>
<td>measure for statistics, e.g. factorization time (0 - off, 1 - user time, 2 - wallclock time)</td>
<td>1</td>
</tr>
<tr>
<td>int:timer</td>
<td>type of timer (1 - cputime, aka. usertime, 2 - wallclock time, 0 - no timing)</td>
<td>2</td>
</tr>
<tr>
<td>int:verbosity</td>
<td>verbosity level (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full)</td>
<td>3</td>
</tr>
<tr>
<td>real:epsilon_factorization</td>
<td>zero tolerance used in factorization</td>
<td>1e-20</td>
</tr>
<tr>
<td>real:epsilon_pivot</td>
<td>pivot zero tolerance used in factorization</td>
<td>1e-10</td>
</tr>
<tr>
<td>real:epsilon_update</td>
<td>zero tolerance used in update of the factorization</td>
<td>1e-16</td>
</tr>
<tr>
<td>real:epsilon_zero</td>
<td>general zero tolerance</td>
<td>1e-16</td>
</tr>
<tr>
<td>real:feastol</td>
<td>primal feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>real:fpfeastol</td>
<td>working tolerance for feasibility in floating-point solver during iterative refinement</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-12, 1]</td>
<td></td>
</tr>
<tr>
<td>real:fpopttol</td>
<td>working tolerance for optimality in floating-point solver during iterative refinement</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-12, 1]</td>
<td></td>
</tr>
<tr>
<td>real:infty</td>
<td>infinity threshold</td>
<td>∞</td>
</tr>
<tr>
<td>real:leastsq_acrcy</td>
<td>accuracy of conjugate gradient method in least squares scaling</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>(higher value leads to more iterations)</td>
<td></td>
</tr>
<tr>
<td>real:liftmaxval</td>
<td>lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated)</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>Range: [10, ∞]</td>
<td></td>
</tr>
<tr>
<td>real:liftminval</td>
<td>lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated)</td>
<td>0.000976562</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 0.1]</td>
<td></td>
</tr>
<tr>
<td>real:maxscaleincr</td>
<td>maximum increase of scaling factors between refinements</td>
<td>1e+25</td>
</tr>
<tr>
<td></td>
<td>Range: [1, ∞]</td>
<td></td>
</tr>
<tr>
<td>real:min_markowitz</td>
<td>minimal Markowitz threshold in LU factorization</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0.0001, 0.9999]</td>
<td></td>
</tr>
<tr>
<td>real:minred</td>
<td>minimal reduction (sum of removed rows/cols) to continue simplification</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>real:objlimit_lower</td>
<td>lower limit on objective value</td>
<td>GAMS cutoff, if maximizing, else -∞</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>real:objlimit_upper</td>
<td>upper limit on objective value</td>
<td>GAMS cutoff, if minimizing, else +∞</td>
</tr>
<tr>
<td></td>
<td>Range: real</td>
<td></td>
</tr>
<tr>
<td>real:opttol</td>
<td>dual feasibility tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>real:refac_basis_nnz</td>
<td>refactor threshold for nonzeros in last factorized basis matrix compared to updated basis matrix</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 100]</td>
<td></td>
</tr>
<tr>
<td>real:refac_mem_factor</td>
<td>refactor threshold for memory growth in factorization since last refactorization</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 10]</td>
<td></td>
</tr>
<tr>
<td>real:refac_update_fill</td>
<td>refactor threshold for fill-in in current factor update compared to fill-in in last factorization</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 100]</td>
<td></td>
</tr>
</tbody>
</table>
### 5.41 XA

#### 5.41.1 Introduction

This document describes the GAMS/XA linear and mixed-integer programming solver. The GAMS/XA solver (here also simply referred to as XA) is based on Sunset Software Technology's XA Callable Library, an implementation of high performance solvers for LP and MIP problems.

XA implements primal simplex, dual simplex, and barrier algorithms for solving linear problems. The primal/dual simplex method is very robust, and in most cases you should get good performance, especially from a warm start. The barrier method is particularly efficient on large models. Both algorithms benefit from XA's presolver, which reduces the size of the model by removing redundant contraints, substituting constraints, etc.

In most cases, GAMS/XA should perform satisfactorily without using any options. However, if fine-tuning is necessary or desired, XA provides many options and parameters designed for this purpose. These options are accessible via GAMS option statements or via an XA-specific option file.

#### 5.41.2 Usage

If you have installed the GAMS system and configured XA as the default LP, RMIP and MIP solver, all LP, RMIP and MIP models without a specific solver option will use XA. If you installed another solver as the default, you can explicitly request that a particular model be solved by XA by inserting the statement

```gams
option LP = xa; { or MIP or RMIP }
```

somewhere before the `solve` statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>real:representation_switch</td>
<td>threshold on number of rows vs. number of columns for switching from column to row representations in auto mode</td>
<td>1.2</td>
</tr>
<tr>
<td>real:simplifier_modifyrowfac</td>
<td>modify constraints when the number of nonzeros or rows is at most this factor times the number of nonzeros or rows before presolving</td>
<td>1</td>
</tr>
<tr>
<td>real:sparsity_threshold</td>
<td>sparse pricing threshold (violations &lt; dimension * SPARSITY_THRESHOLD activates sparse pricing)</td>
<td>0.6</td>
</tr>
<tr>
<td>real:timelimit</td>
<td>time limit in seconds</td>
<td>GAMS reslim</td>
</tr>
</tbody>
</table>
5.41.3 Memory Usage

By default, the GAMS/XA link computes an estimate of the amount of memory that will be required by the solver, and passes this on to the solver. The solver makes an allocation of this amount and then uses this memory during the course of program execution. Usually, this will be sufficient to solve the problem successfully. In some cases, though, the computed estimate will be too small, and GAMS/XA will indicate that a larger memory estimate is required. You will need to manually specify a larger memory estimate to solve the model.

A model-specified memory estimate can be made by adding the following line to your GAMS model before the solve statement:

```
<modelname>.workspace = xx;
```

where xx is the amount of memory in Mbytes. You can also define the environment variable XAMEMORY to be the amount of memory to use, in Mbytes. The computed memory estimate is the default, and is used only if no manual estimate is specified. The model-specified workspace limit overrides the computed estimate, and the XAMEMORY environment variable takes precedence over both of these.

In an attempt to insure that all models solve without running out of memory, XA makes one final memory check and if the user supplied memory amount is below what XA would consider reasonable for that size of problem, XA will then increase your amount to XA's minimal value.

On multi-processor machines, XA will automatically detect and use all available processors (CPU's) when solving MIP models. The memory estimate computed adds 50% more memory per processor to take full advantage of these processors, but this is sometimes not enough memory for XA to multi-process. In this case, a larger estimate must be specified manually.

5.41.4 Semi-Continuous and Semi-Integer Variables

XA supports semi-continuous and semi-integer variable types. Semi-continuous variables are variables that are either at zero or greater than or equal to their lower bound. E.g. a pump motor if operating must run between 2400 and 5400 r.p.m., but it may be switched off as well. Investment levels must exceed a specific threshold or no investment is made.

All semi-continuous variables must have a lower bound specification, e.g., `speed.lo(i) = 100`. Semi-integer variables must have an upper bound as well.

Prior to the introduction of these variable types, semi-continuous variables had to be emulated by adding one additional binary variable and one additional constraint for each semi-continuous variable. For models of any size, this approach very quickly increased the model's size beyond solvability. Now XA has implicitly defined these variables without requiring the addition of new variables and constraints to your model. This effectively increases the size of model that can be solved and does it in a very neat and clean way besides.

For example, to define variables 'a' and 'b' as semi-continuous enter:

```
SemiCont  a ,  b ;
```

or to define semi-integer variables -

```
SemiInt   y1 ,  y2 ;
```

Priority values (.prior suffix) can be associated with both semi-continuous and semi-integer variables. All the integer solving options are available for models with semi-continuous and semi-integer variables as well. For example, you can select solving strategies, `optcr` and `optca` values, etc.

The solve time complexity for semi-continuous variables is comparable with the solve times for binary models, while the semi-integer case compares to integer.
5.41.5 Branch and Bound Topics

XA is designed to solve a vast majority of LP problems using the default settings. In the integer case, however, the default settings may not result in optimal speed and reliability. By experimenting with the control parameters performance can be improved (or worsened!) dramatically.

5.41.5.1 Branching Priorities

Using priorities can significantly reduce the amount of time required to obtain a good integer solution. If your model has a natural order in time, or in space, or in any other dimension then you should consider using priority branching. For example, a multi-period production problem with inventory would use the period value as the priority setting for all variables indexed by period. In a layered chip manufacturing process, the priority assigned to binary variables would be top down or bottom up in that layer.

If priorities are given to binary, integer, or semi-continuous variables, then these are used to provide a user-specified order in which variables are branched. XA selects the variable with the highest priority (lowest numerical value) for branching and the Strategy determines the direction, up or down.

Priorities are assigned to variables using the .prior suffix. For example:

\[
\begin{align*}
\text{NAVY.PRIOROPT} & = 1 ; \\
\ldots \\
\text{Z.PRIOR(J,"SMALL")} & = 10 ; \\
\text{Z.PRIOR(J,"MEDIUM")} & = 5 ; \\
\text{Z.PRIOR(J,"LARGE")} & = 1 ;
\end{align*}
\]

The value 1 indicates the highest priority (branch first), and the value 10 the lowest priority (branch last). Valid priority values should range between \(-32000\) and \(32000\). The default priority value is \(16000\).

5.41.5.2 Branching Strategies

Ten branch & bound strategies are provided to meet the demands of many different types of problems. Each strategy has five variations (six if you include the basic strategy, or "no variation") that affect the solve time, speed to first solution, and the search for the best integer solution. The order in which integer variables are processed during the search is important. This order is called the branching order of integer variables. Solution times can vary significantly with the method selected.

In general, XA will solve your MIP problems much faster when all model variables are given some kind of objective function value. This biases the basis in a specific direction and usually leads to satisfactory first integer solutions.

The strategy used can be changed by setting the strategy option in an XA options file.

<table>
<thead>
<tr>
<th>Branch &amp; Bound Strategy</th>
<th>Description of Selection Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Proprietary method. Default value. Excellent strategy, also add priority to integer variable and try 1P for additional performance gains.</td>
</tr>
<tr>
<td>2</td>
<td>Minimum change in the objective function. This strategy has not been very successful at solving MIP problems.</td>
</tr>
<tr>
<td>3</td>
<td>Priority based upon column order. This strategy probably does not have much meaning because you typically do not set the column order in GAMS</td>
</tr>
</tbody>
</table>
Branch & Bound Strategy | Description of Selection Criteria
--- | ---
4 | Column closest to its integer bound. This strategy tends to send a variable to its lower bound.
6 | Column always branches up (high). Second choice after 1. Excellent choice when your model is a multi-period problem; additional performance gains when priority value is equated with period number; also try 6P if using priorities.
7 | Column always branches down (low). Useful if variable branched down doesn’t limit capacity or resources. One suggestion is to use priorities in the reverse order from that described in Strategy 6.
8 | Column farthest from its integer bound. Next to Strategies 1 and 6 this is probably the next choice to try. Using priorities and variation P is also suggested.
9 | Column randomly selected, useful when solving very large problems. Priority values helpful in multi-period models.
10 | Apparent smoothest sides on the polytope. Priorities helpful.

Each XA B&B strategy has many variations. Sometimes these variations reduce the solution time but may not yield the optimal integer solution. If you are interested in obtaining a fast and 'good' integer solution (which may not be the optimal integer solution), try these variations. You should be aware, though, that using these variations will invalidate the best bound and optimality gap statistics printed by the link at the end of the solve. To choose a variation, either append its letter to the strategy number or add its offset to the strategy number. For example, to choose variations B and P of strategy 6, you could either set "strategy 6BP" or "strategy 1806".

<table>
<thead>
<tr>
<th>Variation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A (+100)</strong></td>
<td>This variation reduces the amount of time XA spends estimating the value of a potential integer solution. The values calculated are rough estimates and may eliminate nodes that would lead to better integer solutions. Variation A may not appear with variation B.</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>This variation spends very little time calculating estimated integer solutions at each node and is the most radical in performance and integer solution value and may eliminate nodes that would lead to better integer solutions. Variation B may not appear with variation A.</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>Each time an improving integer solution is found XA splits the remaining node list in half based upon the length of the current list. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation C may not appear with variation D.</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>Each time an improving integer solution is found XA splits the remaining node list based upon the difference in current projected objective and the best possible objective value divided by two. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation D may not appear with variation C.</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>Each time a node is generated XA calculates the effects of each non-integer on future objective function values, which is calculation intensive. By assigning branching priorities to your integer variables XA will only perform this calculation on the non-integer variables with the lowest branching priority. This frequently reduces the number of calculations. Variation P may appear with any variation, but to be effective you must assign integer branching priorities.</td>
</tr>
</tbody>
</table>

If you wish to improve your solution times, you should experiment with different Strategies to determine which is best for your problems. We have found that Strategies 1 and 6 work quite well. Also try strategies...
As you gain experience with these Strategies you will be able to make an informed choice.

### 5.41.5.3 Limitsearch Parameter

LIMITSEARCH is used to limit the number of nodes to search by implicitly or explicitly stating a bound on the value of an integer solution. The integer solution obtained, if any, will have a functional value no worse than LIMITSEARCH. The next integer solution will have a monotonically improving objective function value until an optimal integer solution is found and if verified.

If you can estimate the objective function value of a good integer solution, you can avoid nodes that lead to worse solutions and, consequently, speed up the search. However, too restrictive a value may lead to no integer solution at all, if an integer solution with an objective value better than LIMITSEARCH value does not exist. If the search terminates with 'NO INTEGER SOLUTION', you must begin the search again with a less restrictive LIMITSEARCH value. The LIMITSEARCH command line parameter has three methods of specifying a lower limit on the objective function.

<table>
<thead>
<tr>
<th>LIMITSEARCH Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>##</td>
<td>Only search for integer solutions between this value and the 'optimal continuous' solution.</td>
</tr>
<tr>
<td>#%</td>
<td>Only search for integer solutions with #% of the 'optimal continuous' solution.</td>
</tr>
<tr>
<td>(#%)</td>
<td>Solve for the integer solution that is within #% of the 'optimal integer solution'. This can reduce the search time significantly, but the reported integer solution may not be the optimal integer solution: it will only be within #% of it. This is similar to the GAMS optcr option, but setting optcr reports the actual gap: this is the recommended way to run GAMS/XA.</td>
</tr>
</tbody>
</table>

### 5.41.6 The XA Option File

The option file is typically called xa.opt. The following line in a GAMS model signals GAMS/XA to use the option file:

```gams
<modelname>.optfile = 1 ;
```

where modelname is the name of the model specified in the model statement. For instance:

```gams
model m /all/ ;
m.optimfile = 1 ;
option LP = XA ;
solve m using LP minimize z ;
```

For more details on using solver option files, see the section on solver option files.

The XA option file allows you to solver-specific options that are not anticipated by GAMS. Where an XA option and a GAMS option both set the same thing, the setting in the XA option file takes precedence. Option file lines beginning with an asterisk * are comment lines. For example:
* Integer solving strategy.
  Strategy 6P
* Write log information to the screen every 5 seconds.
  Set FreqLog 00:05
* Do NOT scale the problem.
  Set Scale No

The contents of the option file are echoed to the screen. If no options file is found where one is expected, a warning is sent to the log file and the solve continues.

Here is a list of available XA options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>basis</td>
<td>controls writing of advanced basis file for restarts</td>
<td>never</td>
</tr>
<tr>
<td></td>
<td>After XA has solved your problem, the solution is saved for the next time the problem is solved. This can greatly reduce the number of iterations and execution time required. By default, the dual simplex algorithm is used when XA detects advance basis restarts. You can instruct XA to not use the dual simplex algorithm for restarts with a setting of set DualSimplex no.</td>
<td></td>
</tr>
<tr>
<td>force</td>
<td>force internal relaxation of bounds if infeasible</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>If your LP model is infeasible, XA makes adjustments in column and row bounds to make the problem feasible. No adjustments are made to binary columns or RHS values of SOS sets. Depending upon how tightly constrained the problem is, XA may be prevented from making additional adjustment that would lead to an integer solution. No adjustments are made to make a column's lower bound less than zero.</td>
<td></td>
</tr>
<tr>
<td>limitsearch</td>
<td>limit searching of the branch-and-bound tree</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>See Section Limitsearch Parameter.</td>
<td></td>
</tr>
<tr>
<td>matlist</td>
<td>display the problem for debugging</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Problem is displayed in equation format. This is probably the most useful command when debugging the model. The GAMS equation and variable listings perform a similar function. Var: List columns in each row Con: List rows in each column Both: Do both Var and Con None: No debugging display</td>
<td></td>
</tr>
<tr>
<td>set barrier</td>
<td>select the primal-dual interior point algorithm</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>The barrier algorithm is most useful when solving large-scale LP models. Yes: Use barrier, with crossover on MIP only No: Use simplex X: Use barrier, with crossover</td>
<td></td>
</tr>
<tr>
<td>set bell</td>
<td>controls the XA termination bell</td>
<td>0</td>
</tr>
<tr>
<td>set bvpriority</td>
<td>set default priority of all binary variables</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td>By default, all discrete variables have priority 1600, so a value &lt; 1600 causes binary variables to be branched on before general integer variables. A value &gt; 1600 has the opposite affect.</td>
<td></td>
</tr>
<tr>
<td>set crash</td>
<td>control basis-crashing method</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0: Minimize primal infeasibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Minimize dual infeasibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2: Both 0 &amp; 1</td>
<td></td>
</tr>
<tr>
<td>set degeniter</td>
<td>consecutive degenerate pivots before anti-cycling code is activated</td>
<td>internal</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>set dualsimplex</td>
<td>controls when to use dual simplex method</td>
<td>auto</td>
</tr>
<tr>
<td></td>
<td>By default, XA chooses when to use dual simplex automatically, e.g. when restarting from an advanced basis.</td>
<td></td>
</tr>
<tr>
<td>set freqlog</td>
<td>time frequency for printing the iteration log</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Frequency in time to print the iteration log line. A negative number (e.g. -00:02) overwrites the same line. This command reduces the overhead of printing too many iteration lines.</td>
<td></td>
</tr>
<tr>
<td>set intgap</td>
<td>Minimum objective improvement between incumbents</td>
<td>GAMS Cheat</td>
</tr>
<tr>
<td></td>
<td>Minimum objective function improvement between each new integer solution. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set intlimit</td>
<td>limit number of incumbents found</td>
<td>GAMS Integer3</td>
</tr>
<tr>
<td></td>
<td>After finding N improving integer solutions, XA terminates with the best solution found thus far. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set intpct</td>
<td>% of available integer columns to consider fixing at each node</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Useful on very large binary problems. If 100 is entered then all integer columns that are integer at the end of solving the relaxed LP problem are fixed at the current integer bounds. Range: [0, 100]</td>
<td></td>
</tr>
<tr>
<td>set iround</td>
<td>control rounding of integer vars when XA reports solutions</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>XA reports either rounded or unrounded integer column primal activity.</td>
<td></td>
</tr>
<tr>
<td>set iteration</td>
<td>sets limit on simplex iterations</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td></td>
<td>XA terminates if limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set ltolerance</td>
<td>lower tolerance for integrality: (i-utol,i+ltol) is integral</td>
<td>5e-6</td>
</tr>
<tr>
<td></td>
<td>The tolerances XA uses to decide that an integer column is integral. For instance, you might consider using a UTOLERANCE of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then 3 * Y is in the range [3 * 3.95, 3 * 4.001].</td>
<td></td>
</tr>
<tr>
<td>set markowitz</td>
<td>Markowitz pivot selection control</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Balances numeric stability vs. sparsity in basis updating and factoring. Larger values favor sparsity over numeric stability.</td>
<td></td>
</tr>
<tr>
<td>set maxcpu</td>
<td>limits the number of threads to use for MIP models</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In general, MIP models should solve faster when running on multiple cores. Consider requesting 50% more memory per thread. This number can be greater than the number of available cores.</td>
<td></td>
</tr>
<tr>
<td>set maxnodes</td>
<td>estimate of the number of nodes in the branch-and-bound tree</td>
<td>internal</td>
</tr>
<tr>
<td>set nodelimit</td>
<td>limits the number of nodes in the branch-and-bound tree</td>
<td>GAMS Nodelim</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>set perturbate</td>
<td>controls perturbation on highly degenerate problems</td>
<td>GAMS Real2</td>
</tr>
<tr>
<td></td>
<td>A positive value allows XA to generate a uniformly distributed random variable between 0 and V. A negative value uses a constant perturbation of -V. Note: This option should not be used except when all else fails. XA has build-in routines to handle degeneracy.</td>
<td></td>
</tr>
<tr>
<td>set pricing</td>
<td>controls the variable pricing strategy</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Useful if XA appears to make a substantial number (rows/2) of pivots that do not move the objective function or reduce the sum of infeasibilities. This feature requires more memory because an additional set of matrix coefficients are loaded into memory. 0: Standard reduced cost pricing 1: Automatic DEVEX pricing switch over 2: Infeasible DEVEX pricing 3: Feasible DEVEX pricing (our next choice) 4: Both infeasible and feasible DEVEX pricing</td>
<td></td>
</tr>
<tr>
<td>set reducedcost</td>
<td>zero tolerance for the dual variables</td>
<td>1e-7</td>
</tr>
<tr>
<td>set reinvertfreq</td>
<td>set basis re-inversion frequency</td>
<td>40</td>
</tr>
<tr>
<td>set relaxed</td>
<td>relax integrality requirement in MIP models - solve as an RMIP</td>
<td>0</td>
</tr>
<tr>
<td>set scale</td>
<td>set scaling method</td>
<td>internal</td>
</tr>
<tr>
<td></td>
<td>2: Row scaling only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Yes: Column and row scaling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No: Data not scaled</td>
<td></td>
</tr>
<tr>
<td>set stickwithit</td>
<td>controls how XA follows branching advice from an integer basis</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>If an integer basis (.b01) file is reloaded to indicate branching direction for the current XA solve, this branching advice is following until N infeasible branches are made. After N infeasible branches, the standard branching direction for the particular branch &amp; bound strategy is used.</td>
<td></td>
</tr>
<tr>
<td>set tctolerance</td>
<td>zero tolerance for technology coefficients</td>
<td>1e-7</td>
</tr>
<tr>
<td></td>
<td>The smallest technological coefficient allowed in your constraint matrix. This tolerance is useful when extensive calculations are performed on these coefficients, where the results should be zero (but because of rounding errors) ends up being something like 1.0e-15.</td>
<td></td>
</tr>
<tr>
<td>set timelimit</td>
<td>set the time limit</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td></td>
<td>Maximum time allowed for solving the problem. XA terminates if this limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Wall clock time is used. If set too low, reported integer solutions may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set tolerance dual</td>
<td>dual feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>set tolerance primal</td>
<td>primal feasibility tolerance</td>
<td>1e-7</td>
</tr>
</tbody>
</table>
5.41 XA

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>set utolerance</td>
<td>upper tolerance for integrality: (i-utol, i+ltol) is integral. The tolerances XA uses to decide that an integer column’s value is integral. For instance, you might consider using a UTOLERANCE of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then 3 * Y is in the range [3 * 3.95, 3 * 4.001].</td>
<td>5e-6</td>
</tr>
<tr>
<td>set ypivot</td>
<td>zero tolerance for dual values when pivoting. When selecting a column to leave the basis, columns with absolute marginal values less than ypivot are rejected. Pivoting in columns with very small values can lead to numeric instability and should be avoided when possible. Setting ypivot too large can lead to infeasible pivoting. Extreme caution should be exercised when changing this value because of the overall effect on problem feasibility.</td>
<td>1e-9</td>
</tr>
<tr>
<td>stopafter</td>
<td>post-integer-feasible time limit for MIP models. Amount of time (hh:mm:ss) to continue solving after finding the first integer solution. Reported integer solution may not be the optimal integer solution because of premature termination. 0 disables this cheat.</td>
<td>GAMS Real3</td>
</tr>
<tr>
<td>strategy</td>
<td>controls the MIP search strategy</td>
<td>GAMS Integer1</td>
</tr>
<tr>
<td>tomps</td>
<td>write an MPS file (gams.mps) for the problem</td>
<td>no</td>
</tr>
</tbody>
</table>

### 5.41.7 Iteration Log Formats

The iteration log is something many users watch closely, especially when solving MIP models. Setting Set FreqLog 0 suppresses the display of the iteration log. During LP iterations, the log format varies depending on the algorithm chosen. Its format is self-explanatory. The default MIP log looks like:

<table>
<thead>
<tr>
<th>Node</th>
<th>IInf</th>
<th>ToGo.Map</th>
<th>Best.Obj</th>
<th>Cur.Obj</th>
<th>Int.Obj</th>
<th># Column +/-</th>
<th>Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>#####</td>
<td>###</td>
<td>########</td>
<td>########</td>
<td></td>
<td></td>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node: Active node, the smaller the better, value increases and decreases as the branch- and-bound proceeds.</td>
</tr>
<tr>
<td>IInf: Number of discrete columns having fractional values. This number converges to 0 as XA approaches an integer solution.</td>
</tr>
<tr>
<td>ToGo.Map: A numeric picture of open nodes. The i’th digit (from the right) represents the number of open nodes in the i’th group of ten nodes. For example, 435 means: - 4 unexplored nodes between nodes 20 and 29. - 3 unexplored nodes between nodes 10 and 19. - 5 unexplored nodes between nodes 0 and 9.</td>
</tr>
<tr>
<td>Best.Obj: Best possible integer objective function. As the branch-and-bound algorithm proceeds this number bounds the Optimal Integer Solution. This number does not change very fast.</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
</tr>
<tr>
<td>Cur.0bj</td>
</tr>
<tr>
<td>Int.0bj</td>
</tr>
<tr>
<td>#</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>+/-</td>
</tr>
<tr>
<td>Iter</td>
</tr>
</tbody>
</table>

Use the Set FreqLog option to minimize the number of lines displayed. Logging each iteration can significantly slow down the solution process.

5.42 XPRESS

5.42.1 Introduction

The GAMS/XPRESS solver is based on the XPRESS Optimization Subroutine Library, and runs only in conjunction with the GAMS modeling system. GAMS/XPRESS (also simply referred to as XPRESS) is a versatile, high-performance optimization system. The system integrates:

- a powerful simplex-based LP solver.
- a MIP module with cut generation for integer programming problems.
- a barrier module implementing a state-of-the-art interior point algorithm for very large LP problems.
- a sequential linear programming solver (SLP) for (mixed-integer) nonlinear programs NLP, CNS and MINLP.

The GAMS/XPRESS solver is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes XPRESS, there is no size or algorithm restriction imposed by the license, nor is any separate licensing procedure required. XPRESS SLP requires a GAMS license that includes XPRESS for continuous nonlinear models or for mixed-integer nonlinear models. In order to use XPRESS Knitro, the GAMS license must include GAMS/Knitro in addition to the XPRESS license for (mixed-integer) nonlinear models.

5.42.2 Usage

To explicitly request that a model be solved with XPRESS, insert the statement

```
option LP = xpress; { or MIP, RMIP, NLP, CNS, DNLP, RMINLP, MINLP, QCP, MIQCP, or RMIQCP }
```
somewhere before the solve statement. If XPRESS has been selected as the default solver (e.g. during GAMS installation) for the model type in question, the above statement is not necessary.

The standard GAMS options (e.g. iterlim, optcr) can be used to control XPRESS. For more details, see section Controlling a Solver via GAMS Options. Please note however that - apart from reslim - these are only used for linear and quadratic programs and not the nonlinear solves. Termination conditions for XPRESS SLP can be set in SLP Termination Options.

In addition, XPRESS-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

An example of a valid XPRESS option file is:

```
* sample XPRESS options file
algorithm simplex
presolve 0
IterLim 50000
```

In general this is enough knowledge to solve your models. In some cases you may want to use some of the XPRESS options to gain further performance improvements or for other reasons.

### 5.42.2.1 Linear and Quadratic Programming

The options advBasis, lpFlags, defaultAlg, basisOut, mpsOutputFile, reform, reRun, and reslim control the behavior of the GAMS/XPRESS link. The options crash, lpIterlimit, presolve, scaling, threads, and trace set XPRESS library control variables, and can be used to fine-tune XPRESS. See section General LP / MIP / QP Options for more details of XPRESS general options.

#### 5.42.2.1.1 LP

See section LP Options for more details of XPRESS library control variables which can be used to fine-tune the XPRESS LP solver.

#### 5.42.2.1.2 MIP

In some cases, the branch-and-bound MIP algorithm will stop with a proven optimal solution or when unboundedness or (integer) infeasibility is detected. In most cases, however, the global search is stopped through one of the generic GAMS options:

1. iterlim (on the cumulative pivot count) or reslim (in seconds of CPU time),
2. optca & optcr (stopping criteria based on gap between best integer solution found and best possible) or
3. nodlim (on the total number of nodes allowed in the B&B tree).

It is also possible to set the maxNode and maxMipSol options to stop the global search: see section MIP Options for XPRESS control variables for MIP. The options loadMipSol, mipCleanup, mipTrace, mipTraceNode, and mipTraceTime control the behavior of the GAMS/XPRESS link on MIP models. The other options in section MIP Options set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.
5.42.2.1.3 MIP Solution Pool  Typically, XPRESS finds a number of integer feasible points during its global search, but only the final solution is available. The MIP solution pool capability makes it possible to store multiple integer feasible points (aka solutions) for later processing. The MIP solution pool operates in one of two modes: by default \( \text{solnpoolPop} = 1 \) the global search is not altered, but with \( \text{solnpoolPop} = 2 \) a selected set (potentially all) of the integer feasible solutions are enumerated.

The MIP enumeration proceeds until all MIP solutions are enumerated or cut off, or until a user-defined limit is reached. Whenever a new solution is generated by the enumerator, it is presented to the solution pool manager. If there is room in the pool, the new solution is added. If the pool is full, a cull round is performed to select a number of solutions to be thrown out - these solutions can be those stored in the pool and/or the new solution. Solutions can be selected for culling based on their MIP objective value and/or the overall diversity of the solutions in the pool. If neither is chosen, a default choice is made to throw out one solution based on objective values. Whenever a solution is thrown out based on its MIP objective, the enumeration space is pruned based on the cutoff defined by this objective value.

By default, the capacity of the pool is set very large, as is the number of cull rounds to perform, so selecting only \( \text{solnpoolPop} = 2 \) will result in full enumeration. However, many different strategies can be executed by setting the solution pool options. For example, to choose the \( N \)-best solutions, simply set the solution pool capacity to \( N \). When the pool is full, new solutions will force a cull round, and the default is to reject one solution based on its objective and update the cutoff accordingly. To generate all solutions with an objective as good as \( X \), leave the pool capacity set at a high level but set the cutoff to \( X \) using the \( \text{mipabsCutoff} \) option. To return the \( N \)-first solutions, set the solution pool capacity to \( N \) and \( \text{solnpoolCullRounds} = 0 \): as soon as the pool is full the enumeration will stop on the cull round limit.

A number of other strategies for controlling the solution pool behavior are possible by combining different options. Several working examples are provided in the GAMS Test Library in models \texttt{xpress03.gms}, \texttt{xpress04.gms}, and \texttt{xpress05.gms}.

See section MIP Solution Pool Options for XPRESS control variables for MIP Solution Pool.

5.42.2.1.4 Newton-Barrier  The barrier method is invoked by default for quadratic problems, and can be selected for linear models by using one of the options

\[
\begin{align*}
\text{algorithm} & \quad \text{barrier} \\
\text{defaultalg} & \quad 4
\end{align*}
\]

The barrier method is likely to use more memory than the simplex method. No warm start is done, so if an advanced basis exists, you may not wish to use the barrier solver.

See section Newton-barrier Options for XPRESS control variables for the Newton-Barriormethod.

5.42.2.2 Nonlinear Programming

XPRESS can solve nonlinear programs of type NLP, CNS and MINLP (and its relaxed version) using the sequential linear programming solver XPRESS SLP or the interior-point / sequential quadratic programming solver XPRESS Knitro. Convexity is not required, but for non-convex programs XPRESS will in general find local optimal solutions only. The XPRESS multistart can be used to increase the likelihood of finding a good solution by starting from many different initial points.

XPRESS SLP solves nonlinear programs by successive linearization of the nonlinearities. These linearizations, which can be controlled by the options in NLP Augmentation and Linearization Options, are solved by the LP or QCP solver. Therefore, XPRESS user options for LP or QCP are also relevant when solving nonlinear programs. Note, that the NLP Preolve is independent from the LP presolve that is executed in each XPRESS SLP iteration.
5.42.2.2.1 Termination  In most cases it is sufficient to control the termination of XPRESS SLP by 
\texttt{xslp.iterLimit}, \texttt{reslim}, \texttt{xslp.validationTarget.k} and \texttt{xslp.ValidationTarget.r} as the latter two automatically 
control the other XPRESS SLP convergence measures on default. More experienced users may want to 
modify the other convergence measures, which group into:

- Strict convergence: Describes the numerical behaviour of convergence in the formal, mathematical 
  sense. User options: \texttt{xslp.cTol}, \texttt{xslp.aTol.a}, \texttt{xslp.aTol.r}.
- Extended convergence: Measures the quality of the linearization, including the effect of changes to 
  the nonlinear terms that contribute to a variable in the linearization: User options: \texttt{xslp.mTol.a}, 
  \texttt{xslp.mTol.r}, \texttt{xslp.iTol.a}, \texttt{xslp.iTol.r}, \texttt{xslp.sTol.a}, \texttt{xslp.sTol.r}.

When each variable has converged in one of the above cases, XPRESS SLP terminates based on the 
following stopping criteria:

-Baseline static objective convergence measure that compares changes in the objective over a given 
number of iterations relative to the average objective value. User options: \texttt{xslp.vCount}, \texttt{xslp.vLimit}, 
\texttt{xslp.vTol.a}, \texttt{xslp.vTol.r}.
-Static objective convergence measure that is applied when there are no unconverged variables in 
  active constraints. User options: \texttt{xslp.oCount}, \texttt{xslp.oTol.a}, \texttt{xslp.oTol.r}.
-Static objective convergence measure that is applied when a practical solution (all variables have 
  converged and there are no active step bounds) has been found. User options: \texttt{xslp.xCount}, 
\texttt{xslp.xLimit}, \texttt{xslp.xTol.a}, \texttt{xslp.xTol.r}.
-Extended convergence continuation that is applied when a practical solution has been found. It 
  checks if it is worth continuing. User options: \texttt{xslp.wCount}, \texttt{xslp.wTol.a}, \texttt{xslp.wTol.r}.

The user option \texttt{slpConvergenceOps} enables/disables the different convergence measures and stopping 
conditions.

5.42.2.2 Output  The output or logging can be controlled by the NLP Log Options. The default 
XPRESS SLP iteration output shows:

- **It**: Iteration number.
- **LP**: The LP status of the linearization (0: optimal; I: infeasible; U: unbounded; X: interrupted)
- **NetObj**: The net objective of the SLP iteration.
- **ValObj**: The original objective function value.
- **ErrorSum**: Sum of the error delta variables. A measure of infeasibility.
- **ErrorCost**: The value of the weighted error delta variables in the objective. A measure of the effort 
  needed to push the model towards feasibility.
- **Validate**: Relative feasibility measure (calculated only if convergence is likely)
- **KKT**: Relative optimality measure (calculated only if convergence is likely)
- **Unconv**: The number of SLP variables that are not converged.
- **Ext**: The number of SLP variables that are converged, but only by extended criteria.
- **Action**: Special actions (0: failed line search; B: enforcing step bounds; E: some infeasible rows 
  were enforced; G: global variables were fixed; P: solution needed polishing, postsolve instability; P!: 
  solution polishing failed; R: penalty error vectors removed; V: feasibility validation induces further 
  iterations; K: optimality validation induces further iterations)
- **T**: Time.
5.42.2.2.3 XPRESS Knitro  Nonlinear programs can also be solved by XPRESS Knitro using the option `xslp_solver`. In this case, the nonlinear program is passed to Knitro after the NLP Presolve. Setting `xslp_solver` to auto will enable XPRESS to choose XPRESS SLP or XPRESS Knitro automatically based on the problem instance. XPRESS Knitro options can be specified in a Knitro solver option file, which needs to be selected in `knitroOptFile`. Note, that XPRESS Knitro does not support all Knitro options. Specified but unsupported options trigger a warning and are then ignored.

For more information about this nonlinear programming solver, see the GAMS/Knitro documentation.

5.42.3 Summary of XPRESS Options

5.42.3.1 General LP / MIP / QP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>advBasis</td>
<td>Use advanced basis provided by GAMS</td>
<td>auto</td>
</tr>
<tr>
<td>algorithm</td>
<td>Choose between simplex and barrier algorithm</td>
<td>simplex</td>
</tr>
<tr>
<td>basisOut</td>
<td>Directs optimizer to output an MPS basis file</td>
<td>none</td>
</tr>
<tr>
<td>clamping</td>
<td>Allows for the adjustment of returned solution values such that they are always within bounds</td>
<td>0</td>
</tr>
<tr>
<td>clamping_dual</td>
<td>Adjust primal slack values to always be within constraint bounds</td>
<td>0</td>
</tr>
<tr>
<td>clamping_primal</td>
<td>Adjust primal solution to always be within primal bounds</td>
<td>0</td>
</tr>
<tr>
<td>clamping_rdj</td>
<td>Adjust reduced costs to always be within dual bounds implied by the primal solution</td>
<td>0</td>
</tr>
<tr>
<td>clamping_slacks</td>
<td>Adjust dual solution to always be within the dual bounds implied by the slacks</td>
<td>0</td>
</tr>
<tr>
<td>cpuTime</td>
<td>How time should be measured when timings are reported in the log and when checking against time limits</td>
<td>0</td>
</tr>
<tr>
<td>extraPresolve</td>
<td>No longer has any effect and will be removed from future releases</td>
<td>0</td>
</tr>
<tr>
<td>inputTol</td>
<td>Tolerance on input values elements</td>
<td>0</td>
</tr>
<tr>
<td>ioTimeout</td>
<td>Maximum number of seconds to wait for an I/O operation before it is cancelled</td>
<td>30</td>
</tr>
<tr>
<td>lpIterLimit</td>
<td>Maximum number of iterations that will be performed by primal simplex or dual simplex before the optimization process terminates</td>
<td>maxint</td>
</tr>
<tr>
<td>lpRefineIterLimit</td>
<td>Simplex iteration limit the solution refiner can spend in attempting to increase the accuracy of an LP solution</td>
<td>auto</td>
</tr>
<tr>
<td>maxScaleFactor</td>
<td>Determines the maximum scaling factor that can be applied during scaling</td>
<td>64</td>
</tr>
<tr>
<td>maxTime</td>
<td>Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time</td>
<td>0</td>
</tr>
<tr>
<td>mpsNameLength</td>
<td>Maximum length of MPS names in characters</td>
<td>0</td>
</tr>
<tr>
<td>mpsOutputFile</td>
<td>Name of MPS output file</td>
<td>none</td>
</tr>
<tr>
<td>numericalEmphasis</td>
<td>How much emphasis to place on numerical stability instead of solve speed</td>
<td>auto</td>
</tr>
<tr>
<td>outputControls</td>
<td>Toggles the printing of all control settings at the beginning of the search</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>outputLog</td>
<td>Controls the level of output produced by the Optimizer during optimization</td>
<td>1</td>
</tr>
<tr>
<td>outputTol</td>
<td>Zero tolerance on print values</td>
<td>1e-05</td>
</tr>
<tr>
<td>qextractalg</td>
<td>quadratic extraction algorithm in GAMS interface</td>
<td>0</td>
</tr>
<tr>
<td>randomSeed</td>
<td>Sets the initial seed to use for the pseudo-random number generator in the Optimizer</td>
<td>1</td>
</tr>
<tr>
<td>refineOps</td>
<td>Specifies when the solution refiner should be executed to reduce solution infeasibilities</td>
<td>19</td>
</tr>
<tr>
<td>refineOps_iterativeRefiner</td>
<td>Apply the iterative refiner to refine the solution</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_LpOptimal</td>
<td>Run the solution refiner on an optimal solution of a non-global problem</td>
<td>1</td>
</tr>
<tr>
<td>refineOps_LpPresolve</td>
<td>Run the solution refiner on an optimal solution before postsolve on a non-global problem</td>
<td>1</td>
</tr>
<tr>
<td>refineOps_mipFixGlobals</td>
<td>Refine MIP solutions such that rounding them keeps the problem feasible when reoptimized</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_mipFixGlobalsTarget</td>
<td>Attempt to refine MIP solutions such that rounding them keeps the problem feasible when reoptimized, but accept integers solutions even if refinement fails</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_mipNodeLp</td>
<td>Run the solution refiner on each node of the MIP search</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_mipSolution</td>
<td>Run the solution refiner when a new solution is found during a global search</td>
<td>1</td>
</tr>
<tr>
<td>refineOps_refinerPrecision</td>
<td>Use higher precision in the iterative refinement</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_refinerUseDual</td>
<td>If set, the iterative refiner will use the dual simplex algorithm</td>
<td>0</td>
</tr>
<tr>
<td>refineOps_refinerUsePrimal</td>
<td>If set, the iterative refiner will use the primal simplex algorithm</td>
<td>0</td>
</tr>
<tr>
<td>reform</td>
<td>Substitute out objective var and equ when possible</td>
<td>1</td>
</tr>
<tr>
<td>reRun</td>
<td>Rerun with primal simplex when not optimal/feasible</td>
<td>0</td>
</tr>
<tr>
<td>reslim</td>
<td>Overrides GAMS reslim option</td>
<td></td>
</tr>
<tr>
<td>trace</td>
<td>Display the infeasibility diagnosis during presolve</td>
<td>0</td>
</tr>
<tr>
<td>writePrtSol</td>
<td>Directs optimizer to output a &quot;printsol&quot; file</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.42.3.2 Hardware Related Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barCores</td>
<td>If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>barOrderThreads</td>
<td>If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton-barrier method</td>
<td>auto</td>
</tr>
<tr>
<td>barThreads</td>
<td>If set to a positive integer it determines the number of threads implemented to run the Newton-barrier algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>cacheSize</td>
<td>Newton Barrier: L2 or L3 (see notes) cache size in kB (kilobytes) of the CPU</td>
<td>auto</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>concurrentThreads</td>
<td>Determines the number of threads used by the concurrent solver</td>
<td>auto</td>
</tr>
<tr>
<td>coresPerCpu</td>
<td>Used to override the detected value of the number of cores on a CPU</td>
<td>auto</td>
</tr>
<tr>
<td>cpuPlatform</td>
<td>Newton Barrier: Selects the AMD or Intel x86 vectorization instruction set that Barrier should run optimized code for</td>
<td>-1</td>
</tr>
<tr>
<td>crossoverThreads</td>
<td>Determines the maximum number of threads that parallel crossover is allowed to use</td>
<td>auto</td>
</tr>
<tr>
<td>dualThreads</td>
<td>Determines the maximum number of threads that dual simplex is allowed to use</td>
<td>auto</td>
</tr>
<tr>
<td>heurThreads</td>
<td>Branch and Bound: Number of threads to dedicate to running heuristics on the root node</td>
<td>0</td>
</tr>
<tr>
<td>l1Cache</td>
<td>Newton barrier: L1 cache size in kB (kilo bytes) of the CPU</td>
<td>auto</td>
</tr>
<tr>
<td>maxMemoryHard</td>
<td>Sets the maximum amount of memory in megabytes the optimizer should allocate</td>
<td>unlimited</td>
</tr>
<tr>
<td>maxMemorySoft</td>
<td>When resourceStrategy is enabled, this control sets the maximum amount of memory in megabytes the optimizer targets to allocate</td>
<td>unlimited</td>
</tr>
<tr>
<td>mipThreads</td>
<td>If set to a positive integer it determines the number of threads implemented to run the parallel MIP code</td>
<td>auto</td>
</tr>
<tr>
<td>relaxTreeMemoryLimit</td>
<td>When the memory used by the branch and bound search tree exceeds the target specified by the treeMemoryLimit control, the optimizer will try to reduce this by writing nodes to the global file</td>
<td>0.1</td>
</tr>
<tr>
<td>resourceStrategy</td>
<td>Controls whether the optimizer is allowed to make nondeterministic decisions if memory is running low in an effort to preserve memory and finish the solve</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>Default number of threads used during optimization</td>
<td>1</td>
</tr>
<tr>
<td>treeCompression</td>
<td>When writing nodes to the global file, the optimizer can try to use data-compression techniques to reduce the size of the global file on disk</td>
<td>2</td>
</tr>
<tr>
<td>treeMemoryLimit</td>
<td>Soft limit, in megabytes, for the amount of memory to use in storing the branch and bound search tree</td>
<td>auto</td>
</tr>
<tr>
<td>treeMemorySavingTarget</td>
<td>When the memory used by the branch-and-bound search tree exceeds the limit specified by the treeMemoryLimit control, the optimizer will try to save memory by writing lower-rated sections of the tree to the global file</td>
<td>0.4</td>
</tr>
</tbody>
</table>

### 5.42.3.3 Presolve Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barPresolveOps</td>
<td>Newton barrier: Controls the Newton-Barrier specific presolve operations</td>
<td>0</td>
</tr>
<tr>
<td>dualize</td>
<td>Whether presolve should form the dual of the problem</td>
<td>auto</td>
</tr>
<tr>
<td>dualizeOps</td>
<td>Bit-vector control for adjusting the behavior when a problem is dualized</td>
<td>1</td>
</tr>
<tr>
<td>elimFillin</td>
<td>Amount of fill-in allowed when performing an elimination in presolve</td>
<td>10</td>
</tr>
<tr>
<td>elimTol</td>
<td>Markowitz tolerance for the elimination phase of the presolve</td>
<td>0.001</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>indLinBigM</strong></td>
<td>During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough</td>
<td>100000</td>
</tr>
<tr>
<td><strong>indPreLinBigM</strong></td>
<td>During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough</td>
<td>100</td>
</tr>
<tr>
<td><strong>lpFolding</strong></td>
<td>Simplex and barrier: Whether to fold an LP problem before solving it</td>
<td>auto</td>
</tr>
<tr>
<td><strong>max IMPLIED Bound</strong></td>
<td>Presolve: When tighter bounds are calculated during MIP preprocessing, only bounds whose absolute value are smaller than max IMPLIED Bound will be applied to the problem</td>
<td>1e+08</td>
</tr>
<tr>
<td><strong>mipPresolve</strong></td>
<td>Branch and Bound: Type of integer processing to be performed</td>
<td>-1</td>
</tr>
<tr>
<td><strong>mipPresolve_allowChangeBounds</strong></td>
<td>If node preprocessing is allowed to change bounds on continuous columns</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_allowTreeRestart</strong></td>
<td>Allow the branch-and-bound tree search to be restarted if it appears to be advantageous</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_dualReductions</strong></td>
<td>Dual reductions will be performed at each node</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_globalCoefTightening</strong></td>
<td>Allow global (non-bound) tightening of the problem during the tree search</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_logicPreprocessing</strong></td>
<td>Primal reductions will be performed at each node</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_objBasedReductions</strong></td>
<td>Objective function will be used to find reductions at each node</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_reducedCostFixing</strong></td>
<td>Reduced cost fixing will be performed at each node</td>
<td>1</td>
</tr>
<tr>
<td><strong>mipPresolve_symmetryReductions</strong></td>
<td>Allow that symmetry is used to presolve the node problem</td>
<td>1</td>
</tr>
<tr>
<td><strong>preAnalyticCenter</strong></td>
<td>Determines if analytic centers should be computed and used for variable fixing and the generation of alternative reduced costs (-1: Auto 0: Off, 1: Fixing, 2: Redcost, 3: Both)</td>
<td>auto</td>
</tr>
<tr>
<td><strong>preBasisRed</strong></td>
<td>Determines if a lattice basis reduction algorithm should be attempted as part of presolve</td>
<td>0</td>
</tr>
<tr>
<td><strong>preBndRedCone</strong></td>
<td>Determines if second order cone constraints should be used for inferring bound reductions on variables when solving a MIP</td>
<td>auto</td>
</tr>
<tr>
<td><strong>preBndRedQuad</strong></td>
<td>Determines if convex quadratic contraints should be used for inferring bound reductions on variables when solving a MIP</td>
<td>auto</td>
</tr>
<tr>
<td><strong>preCliqueStrategy</strong></td>
<td>Determines how much effort to spend on clique covers in presolve</td>
<td>-1</td>
</tr>
<tr>
<td><strong>preCoefElim</strong></td>
<td>Presolve: Specifies whether the optimizer should attempt to recombine constraints in order to reduce the number of non zero coefficients when presolving a mixed integer problem</td>
<td>2</td>
</tr>
<tr>
<td><strong>preComponents</strong></td>
<td>Presolve: Determines whether small independent components should be detected and solved as individual subproblems during root node processing</td>
<td>auto</td>
</tr>
<tr>
<td><strong>preComponentsEffort</strong></td>
<td>Presolve: Adjusts the overall effort for the independent component presolver</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>preConeDecomp</td>
<td>Presolve: Decompose regular and rotated cones with more than two elements and apply Outer Approximation on the resulting components</td>
<td>auto</td>
</tr>
<tr>
<td>preConvertSeparable</td>
<td>Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints</td>
<td>auto</td>
</tr>
<tr>
<td>preDomCol</td>
<td>Presolve: Determines the level of dominated column removal reductions to perform when presolving a mixed integer problem</td>
<td>auto</td>
</tr>
<tr>
<td>preDomRow</td>
<td>Presolve: Determines the level of dominated row removal reductions to perform when presolving a problem</td>
<td>auto</td>
</tr>
<tr>
<td>preDupRow</td>
<td>Presolve: Determines the type of duplicate rows to look for and eliminate when presolving a problem</td>
<td>auto</td>
</tr>
<tr>
<td>preElimQuad</td>
<td>Presolve: Allows for elimination of quadratic variables via doubleton rows</td>
<td>auto</td>
</tr>
<tr>
<td>preFolding</td>
<td>Presolve: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition</td>
<td>auto</td>
</tr>
<tr>
<td>preImplications</td>
<td>Presolve: Determines whether to use implication structures to remove redundant rows</td>
<td>auto</td>
</tr>
<tr>
<td>preLinDep</td>
<td>Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem</td>
<td>auto</td>
</tr>
<tr>
<td>preObjCutDetect</td>
<td>Presolve: Determines whether to check for constraints that are parallel or near parallel to a linear objective function, and which can safely be removed</td>
<td>1</td>
</tr>
<tr>
<td>preProbing</td>
<td>Presolve: Amount of probing to perform on binary variables during presolve</td>
<td>auto</td>
</tr>
<tr>
<td>presolve</td>
<td>Determines whether presolving should be performed prior to starting the main algorithm</td>
<td>1</td>
</tr>
<tr>
<td>presolveMaxGrow</td>
<td>Limit on how much the number of non-zero coefficients is allowed to grow during presolve, specified as a ratio of the number of non-zero coefficients in the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>presolveOps</td>
<td>Specifies the operations which are performed during the presolve</td>
<td>511</td>
</tr>
<tr>
<td>presolveOps_dualReductions</td>
<td>Dual reductions</td>
<td>1</td>
</tr>
<tr>
<td>presolveOps_duplicateColRemoval</td>
<td>Duplicate column removal</td>
<td>1</td>
</tr>
<tr>
<td>presolveOps_duplicateRowRemoval</td>
<td>Duplicate row removal</td>
<td>1</td>
</tr>
<tr>
<td>presolveOps_forcingRowRemoval</td>
<td>Forcing row removal</td>
<td>1</td>
</tr>
<tr>
<td>presolveOps_linDependRowRemoval</td>
<td>Linearly dependant row removal</td>
<td>0</td>
</tr>
<tr>
<td>presolveOps_noAdvIpReductions</td>
<td>No advanced IP reductions</td>
<td>0</td>
</tr>
<tr>
<td>presolveOps_noGlobalDomainChange</td>
<td>No semi-continuous variable detection</td>
<td>0</td>
</tr>
<tr>
<td>presolveOps_noIntVarAndSosDetect</td>
<td>No integer variable and SOS detection</td>
<td>0</td>
</tr>
<tr>
<td>presolveOps_noIntVarEliminations</td>
<td>No eliminations on integers</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.42.3.4 Scaling Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoScaling</td>
<td>Whether the Optimizer should automatically select between different scaling algorithms</td>
<td>auto</td>
</tr>
<tr>
<td>barFreeScale</td>
<td>Defines how the barrier algorithm scales free variables</td>
<td>1e-06</td>
</tr>
<tr>
<td>barObjScale</td>
<td>Defines how the barrier scales the objective</td>
<td>auto</td>
</tr>
<tr>
<td>barRhsScale</td>
<td>Defines how the barrier scales the right hand side</td>
<td>auto</td>
</tr>
<tr>
<td>objScaleFactor</td>
<td>Custom global objective scaling factor, expressed as a power of 2</td>
<td>0</td>
</tr>
<tr>
<td>scaling</td>
<td>Determines how the Optimizer will rescale a model internally before optimization</td>
<td>163</td>
</tr>
<tr>
<td>scaling_beforePresolve</td>
<td>Scale before presolve</td>
<td>0</td>
</tr>
<tr>
<td>scaling_bigM</td>
<td>Treat big-M rows as normal rows</td>
<td>0</td>
</tr>
<tr>
<td>scaling_byMaxElemNotGeoMean</td>
<td>0: scale by geometric mean</td>
<td>1</td>
</tr>
<tr>
<td>scaling_colScaling</td>
<td>Column scaling</td>
<td>1</td>
</tr>
<tr>
<td>scaling_curtisReid</td>
<td>Curtis-Reid</td>
<td>0</td>
</tr>
<tr>
<td>scaling_disableGlobalObjScaling</td>
<td>Do not apply automatic global objective scaling</td>
<td>0</td>
</tr>
<tr>
<td>scaling_ignoreQuadRowPart</td>
<td>Exclude the quadratic part of constraint when calculating scaling factors</td>
<td>0</td>
</tr>
<tr>
<td>scaling_maximum</td>
<td>Maximum</td>
<td>0</td>
</tr>
<tr>
<td>scaling_noAggressiveQScaling</td>
<td>Disable aggressive quadratic scaling</td>
<td>0</td>
</tr>
<tr>
<td>scaling_noScalingColsDown</td>
<td>Do not scale columns down</td>
<td>0</td>
</tr>
<tr>
<td>scaling_noScalingRowsUp</td>
<td>Do not scale rows up</td>
<td>0</td>
</tr>
<tr>
<td>scaling_rhsScaling</td>
<td>RHS scaling</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.42.3.5 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaling_rowScaling</td>
<td>Row scaling</td>
<td>1</td>
</tr>
<tr>
<td>scaling_rowScalingAgain</td>
<td>Row scaling again</td>
<td>0</td>
</tr>
<tr>
<td>scaling_simplexObjScaling</td>
<td>Scale objective function for the simplex method</td>
<td>1</td>
</tr>
<tr>
<td>scaling_slackScaling</td>
<td>Enable explicit linear slack scaling</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>algAfterNetwork</td>
<td>Algorithm to be used for the clean up step after the network simplex solver</td>
<td>auto</td>
</tr>
<tr>
<td>autoPerturb</td>
<td>Simplex: Indicates whether automatic perturbation is performed</td>
<td>auto</td>
</tr>
<tr>
<td>bigM</td>
<td>Infeasibility penalty used if the &quot;Big M&quot; method is implemented</td>
<td>auto</td>
</tr>
<tr>
<td>bigMMETHOD</td>
<td>Simplex: Whether to use the &quot;Big M&quot; method, or the standard phase I (achieving feasibility) and phase II (achieving optimality)</td>
<td>1</td>
</tr>
<tr>
<td>crash</td>
<td>Simplex: Determines the type of crash used when the algorithm begins</td>
<td>2</td>
</tr>
<tr>
<td>defaultAlg</td>
<td>Selects the algorithm that will be used to solve the LP</td>
<td>auto</td>
</tr>
<tr>
<td>dualGradient</td>
<td>Simplex: Dual simplex pricing method</td>
<td>auto</td>
</tr>
<tr>
<td>dualPerturb</td>
<td>Factor by which the problem will be perturbed prior to optimization by dual simplex</td>
<td>auto</td>
</tr>
<tr>
<td>dualStrategy</td>
<td>Bit-vector control specifies the dual simplex strategy</td>
<td>1</td>
</tr>
<tr>
<td>etaTol</td>
<td>Tolerance on eta elements</td>
<td>1e-13</td>
</tr>
<tr>
<td>feasTol</td>
<td>Determines when a solution is treated as feasible</td>
<td>1e-06</td>
</tr>
<tr>
<td>feasTolPerturb</td>
<td>Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes</td>
<td>1e-06</td>
</tr>
<tr>
<td>feasTolTarget</td>
<td>Target feasibility tolerance for the solution refiner</td>
<td>0</td>
</tr>
<tr>
<td>forceParallelDual</td>
<td>Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm</td>
<td>0</td>
</tr>
<tr>
<td>invertFreq</td>
<td>Simplex: Frequency with which the basis will be inverted</td>
<td>auto</td>
</tr>
<tr>
<td>invertMin</td>
<td>Simplex: Minimum number of iterations between full inversions of the basis matrix</td>
<td>3</td>
</tr>
<tr>
<td>lpFlags</td>
<td>Bit-vector control which defines the algorithm for solving an LP problem or the initial LP relaxation of a MIP problem</td>
<td>0</td>
</tr>
<tr>
<td>lpFlags_barrier</td>
<td>Use the barrier method</td>
<td>0</td>
</tr>
<tr>
<td>lpFlags_dual</td>
<td>Use the dual simplex method</td>
<td>0</td>
</tr>
<tr>
<td>lpFlags_network</td>
<td>Use the network simplex method</td>
<td>0</td>
</tr>
<tr>
<td>lpFlags_primal</td>
<td>Use the primal simplex method</td>
<td>0</td>
</tr>
<tr>
<td>lpLog</td>
<td>Simplex: Frequency at which the simplex log is printed</td>
<td>100</td>
</tr>
<tr>
<td>lpLogDelay</td>
<td>Time interval between two LP log lines</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>lpLogStyle</td>
<td>Simplex: Style of the simplex log</td>
<td>1</td>
</tr>
<tr>
<td>lpThreads</td>
<td>Parameter is deprecated and will be removed in a future release</td>
<td>-1</td>
</tr>
<tr>
<td>markowitzTol</td>
<td>Markowitz tolerance used for the factorization of the basis matrix</td>
<td>0.01</td>
</tr>
<tr>
<td>matrixTol</td>
<td>Zero tolerance on matrix elements</td>
<td>1e-09</td>
</tr>
<tr>
<td>netStallLimit</td>
<td>Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex, depending on algAfterNetwork</td>
<td>auto</td>
</tr>
<tr>
<td>optimalityTol</td>
<td>Simplex: Zero tolerance for reduced costs</td>
<td>1e-06</td>
</tr>
<tr>
<td>optimalityTolTarget</td>
<td>Target optimality tolerance for the solution refiner</td>
<td>0</td>
</tr>
<tr>
<td>penalty</td>
<td>Minimum absolute penalty variable coefficient</td>
<td>auto</td>
</tr>
<tr>
<td>pivotTol</td>
<td>Simplex: Zero tolerance for matrix elements</td>
<td>1e-09</td>
</tr>
<tr>
<td>ppFactor</td>
<td>Partial pricing candidate list sizing parameter</td>
<td>1</td>
</tr>
<tr>
<td>pricingAlg</td>
<td>Simplex: Determines the primal simplex pricing method</td>
<td>auto</td>
</tr>
<tr>
<td>primalPerturb</td>
<td>Factor by which the problem will be perturbed prior to optimization by primal simplex</td>
<td>auto</td>
</tr>
<tr>
<td>primalUnshift</td>
<td>Determines whether primal is allowed to call dual to unshift</td>
<td>0</td>
</tr>
<tr>
<td>relPivotTol</td>
<td>Simplex: Minimum size of pivot element relative to largest element in column</td>
<td>1e-06</td>
</tr>
<tr>
<td>sifting</td>
<td>Determines whether to enable sifting algorithm with the dual simplex method</td>
<td>auto</td>
</tr>
<tr>
<td>siftPasses</td>
<td>Determines how quickly we allow to grow the worker problems during the sifting algorithm</td>
<td>4</td>
</tr>
<tr>
<td>siftSwitch</td>
<td>Determines which algorithm to use for solving the subproblems during sifting</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 5.42.3.6 QP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>eigenvalueTol</td>
<td>Quadratic matrix is considered not to be positive semi-definite, if its smallest eigenvalue is smaller than the negative of this value</td>
<td>1e-06</td>
</tr>
<tr>
<td>ifCheckConvexity</td>
<td>Determines if the convexity of the problem is checked before optimization</td>
<td>1</td>
</tr>
<tr>
<td>qSimplexOps</td>
<td>Controls the behavior of the quadratic simplex solvers</td>
<td>0</td>
</tr>
<tr>
<td>quadraticUnshift</td>
<td>Determines whether an extra solution purification step is called after a solution found by the quadratic simplex (either primal or dual)</td>
<td>auto</td>
</tr>
<tr>
<td>repairIndefInitEq</td>
<td>Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.42.3.7 Newton-barrier Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>algAfterCrossover</td>
<td>Algorithm to be used for the final clean up step after the crossover</td>
<td>auto</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>barAlg</td>
<td>Determines which barrier algorithm is to be used to solve the problem</td>
<td>auto</td>
</tr>
<tr>
<td>barCrash</td>
<td>Newton barrier: Determines the type of crash used for the crossover</td>
<td>4</td>
</tr>
<tr>
<td>barDualStop</td>
<td>Newton barrier: Convergence parameter, representing the tolerance for dual infeasibilities</td>
<td>auto</td>
</tr>
<tr>
<td>barFailIterLimit</td>
<td>Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>barGapStop</td>
<td>Newton barrier: Convergence parameter, representing the tolerance for the relative duality gap</td>
<td>auto</td>
</tr>
<tr>
<td>barGapTarget</td>
<td>Newton barrier: Target tolerance for the relative duality gap</td>
<td>auto</td>
</tr>
<tr>
<td>barIndefLimit</td>
<td>Newton Barrier: Limits the number of consecutive indefinite barrier iterations that will be performed</td>
<td>15</td>
</tr>
<tr>
<td>barIterLimit</td>
<td>Newton barrier: Maximum number of iterations</td>
<td>500</td>
</tr>
<tr>
<td>barKernel</td>
<td>Newton barrier: Defines how centrality is weighted in the barrier algorithm</td>
<td>0</td>
</tr>
<tr>
<td>barObjPerturb</td>
<td>Defines how the barrier perturbs the objective</td>
<td>1e-06</td>
</tr>
<tr>
<td>barOrder</td>
<td>Newton barrier: Controls the Cholesky factorization in the Newton-Barrier</td>
<td>auto</td>
</tr>
<tr>
<td>barOutput</td>
<td>Newton barrier: Level of solution output provided</td>
<td>1</td>
</tr>
<tr>
<td>barPerturb</td>
<td>Newton barrier: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties</td>
<td>0</td>
</tr>
<tr>
<td>barPrimalStop</td>
<td>Newton barrier: Convergence parameter, indicating the tolerance for primal infeasibilities</td>
<td>auto</td>
</tr>
<tr>
<td>barRefIter</td>
<td>Newton barrier: After terminating the barrier algorithm, further refinement steps can be performed</td>
<td>0</td>
</tr>
<tr>
<td>barRegularize</td>
<td>Determines how the barrier algorithm applies regularization on the KKT system</td>
<td>-1</td>
</tr>
<tr>
<td>barStart</td>
<td>Newton barrier: Controls the computation of the starting point for the barrier algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>barStepStop</td>
<td>Newton barrier: Convergence parameter, representing the minimal step size</td>
<td>1e-16</td>
</tr>
<tr>
<td>choleskyAlg</td>
<td>Newton barrier: Type of Cholesky factorization used</td>
<td>auto</td>
</tr>
<tr>
<td>choleskyTol</td>
<td>Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm</td>
<td>1e-15</td>
</tr>
<tr>
<td>crossover</td>
<td>Newton barrier: Determines whether the barrier method will cross over to the simplex method when an optimal solution has been found, to provide an end basis and advanced sensitivity analysis information</td>
<td>auto</td>
</tr>
<tr>
<td>crossoverAccuracyTol</td>
<td>Newton barrier: Determines how crossover adjusts the default relative pivot tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td>crossoverIterLimit</td>
<td>Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates</td>
<td>2147483647</td>
</tr>
<tr>
<td>crossoverOps</td>
<td>Newton barrier: Bit vector for adjusting the behavior of the crossover procedure</td>
<td>0</td>
</tr>
<tr>
<td>denseColLimit</td>
<td>Newton barrier: Controls trigger point for special treatment of dense columns in Cholesky factorization</td>
<td>auto</td>
</tr>
</tbody>
</table>
### 5.42.3.8 MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>backTrack</td>
<td>Branch and Bound: Specifies how to select the next node to work on when a full backtrack is performed</td>
<td>3</td>
</tr>
<tr>
<td>backtrackTie</td>
<td>Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed</td>
<td>-1</td>
</tr>
<tr>
<td>branchChoice</td>
<td>Once a global entity has been selected for branching, this control determines which of the branches is solved first</td>
<td>0</td>
</tr>
<tr>
<td>branchDisj</td>
<td>Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search</td>
<td>auto</td>
</tr>
<tr>
<td>branchStructural</td>
<td>Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search</td>
<td>auto</td>
</tr>
<tr>
<td>breadthFirst</td>
<td>Number of nodes to include in the best-first search before switching to the local first search (nodeSelection = 4)</td>
<td>11</td>
</tr>
<tr>
<td>deterministic</td>
<td>Branch and Bound: Specifies whether the parallel MIP search should be deterministic</td>
<td>1</td>
</tr>
<tr>
<td>feasibilityPump</td>
<td>Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node</td>
<td>auto</td>
</tr>
<tr>
<td>fixoptfile</td>
<td>name of option file which is read just before solving the fixed problem</td>
<td></td>
</tr>
<tr>
<td>genConsAbsTransformation</td>
<td>Specifies the reformulation method for absolute value general constraints at the beginning of the search</td>
<td>auto</td>
</tr>
<tr>
<td>genConsDualReductions</td>
<td>Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs</td>
<td>1</td>
</tr>
<tr>
<td>historyCosts</td>
<td>Branch and Bound: How to update the pseudo cost for a global entity when a strong branch or a regular branch is applied</td>
<td>auto</td>
</tr>
<tr>
<td>loadMipSol</td>
<td>Loads a MIP solution (the initial point)</td>
<td>0</td>
</tr>
<tr>
<td>localChoice</td>
<td>Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree</td>
<td>auto</td>
</tr>
<tr>
<td>maxLocalBacktrack</td>
<td>Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node</td>
<td>auto</td>
</tr>
<tr>
<td>maxMipSol</td>
<td>Branch and Bound: Limit on the number of integer solutions to be found by the Optimizer</td>
<td>0</td>
</tr>
<tr>
<td>maxMipTasks</td>
<td>Branch-and-Bound: The maximum number of tasks to run in parallel during a MIP solve</td>
<td>auto</td>
</tr>
<tr>
<td>maxNode</td>
<td>Branch and Bound: Maximum number of nodes that will be explored</td>
<td>maxint</td>
</tr>
<tr>
<td>maxStallTime</td>
<td>Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent</td>
<td>0</td>
</tr>
<tr>
<td>mipAbsCutoff</td>
<td>Branch and Bound: If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to mipAbsCutoff</td>
<td>auto</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>mipAbsStop</td>
<td>Branch and Bound: Absolute tolerance determining whether the global search will continue or not</td>
<td>0</td>
</tr>
<tr>
<td>mipAddCutoff</td>
<td>Branch and Bound: Amount to add to the objective function of the best integer solution found to give the new CURRMIPCUTOFF</td>
<td>0</td>
</tr>
<tr>
<td>mipCleanup</td>
<td>Clean up the MIP solution (round-fix-solve) to get duals</td>
<td>1</td>
</tr>
<tr>
<td>mipComponents</td>
<td>Determines whether disconnected components in a MIP should be solved as separate MIPs</td>
<td>auto</td>
</tr>
<tr>
<td>mipConcurrentNodes</td>
<td>Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled</td>
<td>auto</td>
</tr>
<tr>
<td>mipConcurrentSolves</td>
<td>Branch and Bound: Limits operations to start for a MIP</td>
<td>0</td>
</tr>
<tr>
<td>mipDualReductions</td>
<td>Branch and Bound: Limits operations that can reduce the MIP solution space</td>
<td>1</td>
</tr>
<tr>
<td>mipFracReduce</td>
<td>Branch and Bound: Specifies how often the optimizer should run a heuristic to reduce the number of fractional integer variables in the node LP solutions</td>
<td>auto</td>
</tr>
<tr>
<td>mipKappaFreq</td>
<td>Branch and Bound: Specifies how frequently the basis condition number (also known as kappa) should be calculated during the branch-and-bound search</td>
<td>0</td>
</tr>
<tr>
<td>mipLog</td>
<td>Global print control</td>
<td>-100</td>
</tr>
<tr>
<td>mipRampUp</td>
<td>Controls the strategy used by the parallel MIP solver during the ramp-up phase of a branch-and-bound tree search</td>
<td>auto</td>
</tr>
<tr>
<td>mipRefineIterLimit</td>
<td>Defines an effort limit expressed as simplex iterations for the MIP solution refiner</td>
<td>auto</td>
</tr>
<tr>
<td>mipRelCutoff</td>
<td>Branch and Bound: Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of CURRMIPCUTOFF</td>
<td>0</td>
</tr>
<tr>
<td>mipRelStop</td>
<td>Branch and Bound: Determines when the global search will terminate</td>
<td>0.0001</td>
</tr>
<tr>
<td>mipRestart</td>
<td>Branch and Bound: Controls strategy for in-tree restart</td>
<td>auto</td>
</tr>
<tr>
<td>mipRestartFactor</td>
<td>Branch and Bound: Fine tune initial conditions to trigger an in-tree restart</td>
<td>1</td>
</tr>
<tr>
<td>mipRestartGapThreshold</td>
<td>Branch and Bound: Initial gap threshold to delay in-tree restart</td>
<td>0.02</td>
</tr>
<tr>
<td>mipstopexpr</td>
<td>Stopping expression for branch and bound</td>
<td></td>
</tr>
<tr>
<td>mipTol</td>
<td>Branch and Bound: ‘Tolerance within which a decision variable’s value is considered to be integral</td>
<td>5e-06</td>
</tr>
<tr>
<td>mipTolTarget</td>
<td>Target mipTol value used by the automatic MIP solution refiner as defined by refineOps</td>
<td>0</td>
</tr>
<tr>
<td>mipTrace</td>
<td>Name of MIP trace file</td>
<td>none</td>
</tr>
<tr>
<td>mipTraceNode</td>
<td>Node interval between MIP trace file entries</td>
<td>100</td>
</tr>
<tr>
<td>mipTraceTime</td>
<td>Time interval, in seconds, between MIP trace file entries</td>
<td>5</td>
</tr>
<tr>
<td>miqcpAlg</td>
<td>Determines which algorithm is to be used to solve mixed integer quadratic constrained and mixed integer second order cone problems</td>
<td>auto</td>
</tr>
<tr>
<td>nodeProbingEffort</td>
<td>Adjusts the overall level of node probing</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nodeSelection</td>
<td>Branch and Bound: Determines which nodes will be considered for solution once the current node has been solved</td>
<td>auto</td>
</tr>
<tr>
<td>objGoodEnough</td>
<td>Stop once an objective this good is found</td>
<td>none</td>
</tr>
<tr>
<td>pseudoCost</td>
<td>Branch and Bound: Default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search</td>
<td>0.01</td>
</tr>
<tr>
<td>qeRootAlg</td>
<td>Determines which algorithm is to be used to solve the root of a mixed integer quadratic constrained or mixed integer second order cone problem, when outer approximation is used</td>
<td>auto</td>
</tr>
<tr>
<td>sbBest</td>
<td>Number of infeasible global entities to initialize pseudo costs for on each node</td>
<td>auto</td>
</tr>
<tr>
<td>sbEffort</td>
<td>Adjusts the overall amount of effort when using strong branching to select an infeasible global entity to branch on</td>
<td>1</td>
</tr>
<tr>
<td>sbEstimate</td>
<td>Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible global entity to branch on</td>
<td>auto</td>
</tr>
<tr>
<td>sbIterLimit</td>
<td>Number of dual iterations to perform the strong branching for each entity</td>
<td>auto</td>
</tr>
<tr>
<td>sbSelect</td>
<td>Size of the candidate list of global entities for strong branching</td>
<td>-2</td>
</tr>
<tr>
<td>sleepOnThreadWait</td>
<td>Determines if the threads should be put into a wait state when waiting for work</td>
<td>auto</td>
</tr>
<tr>
<td>sosRefTol</td>
<td>Minimum relative gap between the ordering values of elements in a special ordered set</td>
<td>1e-06</td>
</tr>
<tr>
<td>symmetry</td>
<td>Adjusts the overall amount of effort for symmetry detection</td>
<td>1</td>
</tr>
<tr>
<td>symSelect</td>
<td>Adjusts the overall amount of effort for symmetry detection</td>
<td>-1</td>
</tr>
<tr>
<td>varSelection</td>
<td>Branch and Bound: Determines the formula used to calculate the estimate of each integer variable, and thus which integer variable is selected to be branched on at a given node</td>
<td>auto</td>
</tr>
</tbody>
</table>

### 5.42.3.9 MIP Cuts Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoCutting</td>
<td>Automatically decide whether to generate cutting planes at local nodes in the tree</td>
<td>auto</td>
</tr>
<tr>
<td>conflictCuts</td>
<td>Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts</td>
<td>auto</td>
</tr>
<tr>
<td>coverCuts</td>
<td>Branch and Bound: Number of rounds of lifted cover inequalities at the top node</td>
<td>auto</td>
</tr>
<tr>
<td>cutDepth</td>
<td>Branch and Bound: Sets the maximum depth in the tree search at which cuts will be generated</td>
<td>auto</td>
</tr>
<tr>
<td>cutFactor</td>
<td>Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during global search</td>
<td>auto</td>
</tr>
<tr>
<td>cutFreq</td>
<td>Branch and Bound: Frequency at which cuts are generated in the tree search</td>
<td>auto</td>
</tr>
<tr>
<td>cutSelect</td>
<td>Bit vector providing detailed control of the cuts created for the root node of a global solve</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>cutSelect_clique</td>
<td>Clique cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_cover</td>
<td>Lifted cover cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_disableCutRows</td>
<td>Disable cutting from cut rows</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_flowpath</td>
<td>Flow path cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_gubCover</td>
<td>Lifted GUB cover cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_implication</td>
<td>Implication cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_indicator</td>
<td>Indicator constraint cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_liftAndProject</td>
<td>Turn on automatic Lift-and-Project cutting strategy</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_mir</td>
<td>Mixed Integer Rounding (MIR) cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_mirRowAggregation</td>
<td>Turn on row aggregation for MIR cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutSelect_zeroHalf</td>
<td>Zero-half cuts</td>
<td>1</td>
</tr>
<tr>
<td>cutStrategy</td>
<td>Branch and Bound: Cut strategy</td>
<td>auto</td>
</tr>
<tr>
<td>gomCuts</td>
<td>Branch and Bound: Number of rounds of Gomory or lift-and-project cuts at the top node</td>
<td>auto</td>
</tr>
<tr>
<td>inpBest</td>
<td>Number of infeasible global entities to create lift-and-project cuts for during each round of Gomory cuts at the top node (see gomCuts)</td>
<td>50</td>
</tr>
<tr>
<td>inpIterLimit</td>
<td>Number of iterations to perform in improving each lift-and-project cut</td>
<td>auto</td>
</tr>
<tr>
<td>maxCutTime</td>
<td>Maximum amount of time allowed for generation of cutting planes and reoptimization</td>
<td>0</td>
</tr>
<tr>
<td>netCuts</td>
<td>Determines the addition of multi-commodity network cuts to a problem</td>
<td>0</td>
</tr>
<tr>
<td>qCuts</td>
<td>Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation</td>
<td>auto</td>
</tr>
<tr>
<td>treeCoverCuts</td>
<td>Branch and Bound: Number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree</td>
<td>auto</td>
</tr>
<tr>
<td>treeCutSelect</td>
<td>Bit vector providing detailed control of the cuts created during the tree search of a global solve</td>
<td>-1</td>
</tr>
<tr>
<td>treeCutSelect_clique</td>
<td>Clique cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_cover</td>
<td>Lifted cover cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_disableCutRows</td>
<td>Disable cutting from cut rows</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_flowpath</td>
<td>Flow path cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_gubCover</td>
<td>Lifted GUB cover cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_implication</td>
<td>Implication cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_indicator</td>
<td>Indicator constraint cuts</td>
<td>1</td>
</tr>
<tr>
<td>treeCutSelect_liftAndProject</td>
<td>Turn on automatic Lift and Project cutting strategy</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.42.3.10 MIP Heuristics Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heurBeforeLp</td>
<td>Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved</td>
<td>auto</td>
</tr>
<tr>
<td>heurDiveIterLimit</td>
<td>Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic</td>
<td>auto</td>
</tr>
<tr>
<td>heurDiveRandomize</td>
<td>Level of randomization to apply in the diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>heurDiveSoftRounding</td>
<td>Branch and Bound: Enables a more cautious strategy for the diving heuristic, where it tries to push binaries and integer variables to their bounds using the objective, instead of directly fixing them</td>
<td>auto</td>
</tr>
<tr>
<td>heurDiveSpeedUp</td>
<td>Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed</td>
<td>-1</td>
</tr>
<tr>
<td>heurDiveStrategy</td>
<td>Branch and Bound: Chooses the strategy for the diving heuristic</td>
<td>auto</td>
</tr>
<tr>
<td>heurEmphasis</td>
<td>Branch and Bound: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap</td>
<td>-1</td>
</tr>
<tr>
<td>heurForceSpecialObj</td>
<td>Branch and Bound: Whether local search heuristics without objective or with an auxiliary objective should always be used, despite the automatic selection of the Optimizer</td>
<td>0</td>
</tr>
<tr>
<td>heurFreq</td>
<td>Branch and Bound: Frequency at which heuristics are used in the tree search</td>
<td>-1</td>
</tr>
<tr>
<td>heurSearchEffort</td>
<td>Adjusts the overall level of the local search heuristics</td>
<td>1</td>
</tr>
<tr>
<td>heurSearchFreq</td>
<td>Branch and Bound: How often the local search heuristic should be run in the tree</td>
<td>auto</td>
</tr>
<tr>
<td>heurSearchRootCutFreq</td>
<td>How frequently to run the local search heuristic during root cutting</td>
<td>auto</td>
</tr>
<tr>
<td>heurSearchRootSelect</td>
<td>Bit vector control for selecting which local search heuristics to apply on the root node of a global solve</td>
<td>117</td>
</tr>
<tr>
<td>heurSearchTreeSelect</td>
<td>Bit vector control for selecting which local search heuristics to apply during the tree search of a global solve</td>
<td>17</td>
</tr>
</tbody>
</table>

### 5.42.3.11 MIP Solution Pool Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>solnpool</td>
<td>Solution pool file name</td>
<td>none</td>
</tr>
<tr>
<td>solnpoolCapacity</td>
<td>Limit on number of solutions to store</td>
<td>999999999</td>
</tr>
<tr>
<td>solnpoolCullDiversity</td>
<td>Cull N solutions based on solution diversity</td>
<td>-1</td>
</tr>
<tr>
<td>solnpoolCullObj</td>
<td>Cull N solutions based on objective values</td>
<td>-1</td>
</tr>
<tr>
<td>solnpoolCullRounds</td>
<td>Terminate solution generation after N culling rounds</td>
<td>999999999</td>
</tr>
<tr>
<td>solnpoolDupPolicy</td>
<td>Policy to use when handling storage of duplicate solutions</td>
<td>0</td>
</tr>
<tr>
<td>solnpoolmerge</td>
<td>Solution pool file name for merged solutions</td>
<td>none</td>
</tr>
<tr>
<td>solnpoolnumsym</td>
<td>Maximum number of variable symbols when writing merged solutions</td>
<td>10</td>
</tr>
<tr>
<td>solnpoolPop</td>
<td>Controls method used to populate the solution pool</td>
<td>1</td>
</tr>
<tr>
<td>solnpoolPrefix</td>
<td>File name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>solnpoolVerbosity</td>
<td>Controls verbosity of solution pool routines</td>
<td>0</td>
</tr>
</tbody>
</table>

5.42.3.12 General NLP / MINLP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>knitroOptFile</td>
<td>Option file for NLP solver KNITRO</td>
<td></td>
</tr>
<tr>
<td>xslp_algorithm</td>
<td>Bit map describing the SLP algorithm(s) to be used</td>
<td>166</td>
</tr>
<tr>
<td>xslp_algorithm_cascadeBounds</td>
<td>Step bounds are updated to accomodate cascaded values (otherwise cascaded values are pushed to respect step bounds)</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_clampExtendedActiveSB</td>
<td>Apply clamping when converged on extended criteria only with some variables having active step bounds</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_clampExtendedAll</td>
<td>Apply clamping when converged on extended criteria only</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_dynamicDamping</td>
<td>Use dynamic damping</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_escalatePenalties</td>
<td>Escalate penalties</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_estimateStepBounds</td>
<td>Estimate step bounds from early SLP iterations</td>
<td>1</td>
</tr>
<tr>
<td>xslp_algorithm_holdValues</td>
<td>Do not update values which are converged within strict tolerance</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_maxCostOption</td>
<td>Continue optimizing after penalty cost reaches maximum</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_noLPPolishing</td>
<td>Skip the solution polishing step if the LP postsolve returns a slightly infeasible, but claimed optimal solution</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_noStepBounds</td>
<td>Do not apply step bounds</td>
<td>0</td>
</tr>
<tr>
<td>xslp_algorithm_quickConvergenceCheck</td>
<td>Quick convergence check</td>
<td>1</td>
</tr>
<tr>
<td>xslp_algorithm_resetDeltaZ</td>
<td>Reset xslp_delta_z to zero when converged and continue SLP</td>
<td>0</td>
</tr>
</tbody>
</table>
### NLP Presolve Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_linQuadBR</td>
<td>Use linear and quadratic constraints and objective function to further reduce bounds on all variables</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_postsolve</td>
<td>Determines whether postsolving should be performed automatically</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolve</td>
<td>Determines whether presolving should be performed prior to starting the main algorithm</td>
<td>1</td>
</tr>
<tr>
<td>xslp_presolveLevel</td>
<td>Determines the level of changes presolve may carry out on the problem</td>
<td>4</td>
</tr>
<tr>
<td>xslp_presolveOps</td>
<td>Bitmap indicating the SLP presolve actions to be taken</td>
<td>2104</td>
</tr>
<tr>
<td>xslp_presolveOps_domain</td>
<td>Bound tightening based on function domains</td>
<td>1</td>
</tr>
<tr>
<td>xslp_presolveOps_eliminations</td>
<td>Allow eliminations on determined variables</td>
<td>1</td>
</tr>
<tr>
<td>xslp_presolveOps_fixAll</td>
<td>Explicitly fix all columns identified as fixed</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>xslp_presolveOps_fixZero</td>
<td>Explicitly fix columns identified as fixed to zero</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_general</td>
<td>Generic SLP presolve</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_intBounds</td>
<td>MISLP bound tightening</td>
<td>1</td>
</tr>
<tr>
<td>xslp_presolveOps_noCoefficients</td>
<td>Do not presolve coefficients</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_noDeltas</td>
<td>Do not remove delta variables</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_noDualSide</td>
<td>Avoid reductions that can not be dual postsolved</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_noLinear</td>
<td>Avoid performing linear reductions at the nlp level</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_noSimplifier</td>
<td>Avoid simplifying nonlinear expressions</td>
<td>0</td>
</tr>
<tr>
<td>xslp_presolveOps_setBounds</td>
<td>SLP bound tightening</td>
<td>1</td>
</tr>
<tr>
<td>xslp_presolveZero</td>
<td>Minimum absolute value for a variable which is identified as nonzero during SLP presolve</td>
<td>1e-09</td>
</tr>
<tr>
<td>xslp_probing</td>
<td>Determines whether probing on a subset of variables should be performed prior to starting the main algorithm</td>
<td>auto</td>
</tr>
</tbody>
</table>

### 5.42.3.14 NLP Augmentation and Linearization Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_augmentation</td>
<td>Bit map describing the SLP augmentation method(s) to be used</td>
<td>12</td>
</tr>
<tr>
<td>xslp_augmentation_allErrorVectors</td>
<td>Penalty error vectors on all non-linear inequality constraints</td>
<td>1</td>
</tr>
<tr>
<td>xslp_augmentation_allRowErrorVectors</td>
<td>Penalty error vectors on all constraints</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_aMeanWeight</td>
<td>Use arithmetic means to estimate penalty weights</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation EqualityErrorVectors</td>
<td>Penalty error vectors on all non-linear equality constraints</td>
<td>1</td>
</tr>
<tr>
<td>xslp_augmentation_evenHanded</td>
<td>Even handed augmentation</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_minimum</td>
<td>Minimum augmentation</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_noUpdateIfOnlyIV</td>
<td>Initial values do not imply an SLP variable</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_penaltyDeltaVectors</td>
<td>Penalty vectors to exceed step bounds</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_sbFromAbsValues</td>
<td>Estimate step bounds from absolute values of row coefficients</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_sbFromValues</td>
<td>Estimate step bounds from values of row coefficients</td>
<td>0</td>
</tr>
<tr>
<td>xslp_augmentation_stepBoundRows</td>
<td>Row-based step bounds</td>
<td>0</td>
</tr>
<tr>
<td>xslp_delta_x</td>
<td>Minimum absolute value of delta coefficients to be retained</td>
<td>1e-06</td>
</tr>
<tr>
<td>xslp_feasTolTarget</td>
<td>When set, this defines a target feasibility tolerance to which the linearizations are solved to</td>
<td>not set</td>
</tr>
<tr>
<td>xslp_optimalityTolTarget</td>
<td>When set, this defines a target optimality tolerance to which the linearizations are solved to</td>
<td>not set</td>
</tr>
</tbody>
</table>
### 5.42.3.15 NLP Barrier Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_barCrossOverStart</td>
<td>Default crossover activation behaviour for barrier start</td>
<td>0</td>
</tr>
<tr>
<td>xslp_barLimit</td>
<td>Number of initial SLP iterations using the barrier method</td>
<td>0</td>
</tr>
<tr>
<td>xslp_barStallingLimit</td>
<td>Number of iterations to allow numerical failures in barrier before switching to dual</td>
<td>3</td>
</tr>
<tr>
<td>xslp_barStallingObjLimit</td>
<td>Number of iterations over which to measure the objective change for barrier iterations with no crossover</td>
<td>3</td>
</tr>
<tr>
<td>xslp_barStallingTol</td>
<td>Required change in the objective when progress is measured in barrier iterations without crossover</td>
<td>0.05</td>
</tr>
<tr>
<td>xslp_barStartOps</td>
<td>Controls behaviour when the barrier is used to solve the linearizations</td>
<td>-1</td>
</tr>
<tr>
<td>xslp_barStartOps_allowInteriorSol</td>
<td>If a non-vertex converged solution found by barrier without crossover can be returned as a final solution</td>
<td>1</td>
</tr>
<tr>
<td>xslp_barStartOps_stallingNumerical</td>
<td>Fall back to dual simplex if too many numerical problems are reported by the barrier</td>
<td>1</td>
</tr>
<tr>
<td>xslp_barStartOps_stallingObjective</td>
<td>Check objective progress when no crossover is applied</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.42.3.16 NLP Penalty Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_deltaCost</td>
<td>Initial penalty cost multiplier for penalty delta vectors</td>
<td>200</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>xslp_deltaCostFactor</td>
<td>Factor for increasing cost multiplier on total penalty delta vectors</td>
<td>1.3</td>
</tr>
<tr>
<td>xslp_deltaMaxCost</td>
<td>Maximum penalty cost multiplier for penalty delta vectors</td>
<td>1e+20</td>
</tr>
<tr>
<td>xslp_enforceCostShrink</td>
<td>Factor by which to decrease the current penalty multiplier when enforcing rows</td>
<td>1e-05</td>
</tr>
<tr>
<td>xslp_enforceMaxCost</td>
<td>Maximum penalty cost in the objective before enforcing most violating rows</td>
<td>1e+11</td>
</tr>
<tr>
<td>xslp_errorCost</td>
<td>Initial penalty cost multiplier for penalty error vectors</td>
<td>200</td>
</tr>
<tr>
<td>xslp_errorCostFactor</td>
<td>Factor for increasing cost multiplier on total penalty error vectors</td>
<td>1.3</td>
</tr>
<tr>
<td>xslp_errorMaxCost</td>
<td>Maximum penalty cost multiplier for penalty error vectors</td>
<td>1e+20</td>
</tr>
<tr>
<td>xslp_errorTol_a</td>
<td>Absolute tolerance for error vectors</td>
<td>1e-05</td>
</tr>
<tr>
<td>xslp_errorTol_p</td>
<td>Absolute tolerance for printing error vectors</td>
<td>0.0001</td>
</tr>
<tr>
<td>xslp_escalation</td>
<td>Factor for increasing cost multiplier on individual penalty error vectors</td>
<td>1.25</td>
</tr>
<tr>
<td>xslp_eTol_a</td>
<td>Absolute tolerance on penalty vectors</td>
<td>0.0001</td>
</tr>
<tr>
<td>xslp_eTol_r</td>
<td>Relative tolerance on penalty vectors</td>
<td>0.0001</td>
</tr>
<tr>
<td>xslp_evTol_a</td>
<td>Absolute tolerance on total penalty costs</td>
<td>-1</td>
</tr>
<tr>
<td>xslp_evTol_r</td>
<td>Relative tolerance on total penalty costs</td>
<td>-1</td>
</tr>
<tr>
<td>xslp_granularity</td>
<td>Base for calculating penalty costs</td>
<td>4</td>
</tr>
<tr>
<td>xslp_maxWeight</td>
<td>Maximum penalty weight for delta or error vectors</td>
<td>100</td>
</tr>
<tr>
<td>xslp_minWeight</td>
<td>Minimum penalty weight for delta or error vectors</td>
<td>0.01</td>
</tr>
<tr>
<td>xslp_objToPenaltyCost</td>
<td>Factor to estimate initial penalty costs from objective function</td>
<td>0</td>
</tr>
<tr>
<td>xslp_penaltyInfoStart</td>
<td>Iteration from which to record row penalty information</td>
<td>3</td>
</tr>
</tbody>
</table>

### 5.42.3.17 NLP Step Bounds Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_clampShrink</td>
<td>Shrink ratio used to impose strict convergence on variables converged in extended criteria only</td>
<td>0.3</td>
</tr>
<tr>
<td>xslp_clampValidationTol_a</td>
<td>Absolute validation tolerance for applying xslp_clampShrink</td>
<td>not set</td>
</tr>
<tr>
<td>xslp_clampValidationTol_r</td>
<td>Relative validation tolerance for applying xslp_clampShrink</td>
<td>not set</td>
</tr>
<tr>
<td>xslp_defaultStepBound</td>
<td>Minimum initial value for the step bound of an SLP variable if none is explicitly given</td>
<td>16</td>
</tr>
<tr>
<td>xslp_djTol</td>
<td>Tolerance on DJ value for determining if a variable is at its step bound</td>
<td>1e-06</td>
</tr>
<tr>
<td>xslp_expand</td>
<td>Multiplier to increase a step bound</td>
<td>2</td>
</tr>
<tr>
<td>xslp_minSBFactor</td>
<td>Factor by which step bounds can be decreased beneath xslp_eTol_a</td>
<td>1</td>
</tr>
<tr>
<td>xslp_sameCount</td>
<td>Number of steps reaching the step bound in the same direction before step bounds are increased</td>
<td>3</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>xslp_sbStart</td>
<td>SLP iteration after which step bounds are first applied</td>
<td>8</td>
</tr>
<tr>
<td>xslp_shrink</td>
<td>Multiplier to reduce a step bound</td>
<td>0.5</td>
</tr>
<tr>
<td>xslp_shrinkBias</td>
<td>Defines an overwrite / adjustment of step bounds for improving iterations</td>
<td>not set</td>
</tr>
</tbody>
</table>

### 5.42.3.18 NLP Variable Update Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_damp</td>
<td>Damping factor for updating values of variables</td>
<td>1</td>
</tr>
<tr>
<td>xslp_dampExpand</td>
<td>Multiplier to increase damping factor during dynamic damping</td>
<td>1</td>
</tr>
<tr>
<td>xslp_dampMax</td>
<td>Maximum value for the damping factor of a variable during dynamic damping</td>
<td>1</td>
</tr>
<tr>
<td>xslp_dampMin</td>
<td>Minimum value for the damping factor of a variable during dynamic damping</td>
<td>1</td>
</tr>
<tr>
<td>xslp_dampShrink</td>
<td>Multiplier to decrease damping factor during dynamic damping</td>
<td>1</td>
</tr>
<tr>
<td>xslp_dampStart</td>
<td>SLP iteration at which damping is activated</td>
<td>0</td>
</tr>
<tr>
<td>xslp_lsIterLimit</td>
<td>Number of iterations in the line search</td>
<td>0</td>
</tr>
<tr>
<td>xslp_lsPatternLimit</td>
<td>Number of iterations in the pattern search preceding the line search</td>
<td>0</td>
</tr>
<tr>
<td>xslp_lsStart</td>
<td>Iteration in which to active the line search</td>
<td>8</td>
</tr>
<tr>
<td>xslp_lsZeroLimit</td>
<td>Maximum number of zero length line search steps before line search is deactivated</td>
<td>5</td>
</tr>
<tr>
<td>xslpMeritLambda</td>
<td>Factor by which the net objective is taken into account in the merit function</td>
<td>0</td>
</tr>
<tr>
<td>xslp_sameDamp</td>
<td>Number of steps in same direction before damping factor is increased</td>
<td>3</td>
</tr>
</tbody>
</table>

### 5.42.3.19 NLP Termination Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_aTol_a</td>
<td>Absolute delta convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_aTol_r</td>
<td>Relative delta convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_convergenceOps</td>
<td>Bit map describing which convergence tests should be carried out</td>
<td>7167</td>
</tr>
<tr>
<td>xslp_convergenceOps_aTol</td>
<td>Execute the delta tolerance checks</td>
<td>1</td>
</tr>
<tr>
<td>xslp_convergenceOps_cTol</td>
<td>Execute the closure tolerance checks</td>
<td>1</td>
</tr>
<tr>
<td>xslp_convergenceOps_extendedScaling</td>
<td>Take scaling of individual variables / rows into account</td>
<td>0</td>
</tr>
<tr>
<td>xslp_convergenceOps_iTol</td>
<td>Execute the impact tolerance checks</td>
<td>1</td>
</tr>
<tr>
<td>xslp_convergenceOps_mTol</td>
<td>Execute the matrix tolerance checks</td>
<td>1</td>
</tr>
<tr>
<td>xslp_convergenceOps_oTol</td>
<td>Execute the objective range + active step bound check</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_sTol</code></td>
<td>Execute the slack impact tolerance checks</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_validation</code></td>
<td>Execute the validation target convergence checks</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_validationK</code></td>
<td>Execute the first order optimality target convergence checks</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_vTol</code></td>
<td>Execute the objective range checks</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_wTol</code></td>
<td>Execute the convergence continuation check</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_convergenceOps_xTol</code></td>
<td>Execute the objective range + constraint activity check</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_cTol</code></td>
<td>Closure convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_ecfCheck</code></td>
<td>Check feasibility at the point of linearization for extended convergence criteria</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_ecfTol_a</code></td>
<td>Absolute tolerance on testing feasibility at the point of linearization</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_ecfTol_r</code></td>
<td>Relative tolerance on testing feasibility at the point of linearization</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_infeasLimit</code></td>
<td>Maximum number of consecutive infeasible SLP iterations which can occur before Xpress-SLP terminates</td>
<td>3</td>
</tr>
<tr>
<td><code>xslp_iterLimit</code></td>
<td>Maximum number of SLP iterations</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_iTol_a</code></td>
<td>Absolute impact convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_iTol_r</code></td>
<td>Relative impact convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_mTol_a</code></td>
<td>Absolute effective matrix element convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_mTol_r</code></td>
<td>Relative effective matrix element convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_mvTol</code></td>
<td>Marginal value tolerance for determining if a constraint is slack</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_oCount</code></td>
<td>Number of SLP iterations over which to measure objective function variation for static objective (2) convergence criterion</td>
<td>5</td>
</tr>
<tr>
<td><code>xslp_oTol_a</code></td>
<td>Absolute static objective (2) convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_oTol_r</code></td>
<td>Relative static objective (2) convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_sTol_a</code></td>
<td>Absolute slack convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_sTol_r</code></td>
<td>Relative slack convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_stopOutOfRange</code></td>
<td>Stop optimization and return error code if internal function argument is out of range</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_validationTarget_k</code></td>
<td>Optimality target tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td><code>xslp_validationTarget_r</code></td>
<td>Feasibility target tolerance</td>
<td>1e-06</td>
</tr>
<tr>
<td><code>xslp_vCount</code></td>
<td>Number of SLP iterations over which to measure static objective (3) convergence</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_vLimit</code></td>
<td>Number of SLP iterations after which static objective (3) convergence testing starts</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_vTol_a</code></td>
<td>Absolute static objective (3) convergence tolerance</td>
<td>auto</td>
</tr>
</tbody>
</table>
### 5.42 XPRESS

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_vTol_r</td>
<td>Relative static objective (3) convergence tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_wCount</td>
<td>Number of SLP iterations over which to measure the objective for the extended convergence continuation criterion</td>
<td>0</td>
</tr>
<tr>
<td>xslp_wTol_a</td>
<td>Absolute extended convergence continuation tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_wTol_r</td>
<td>Relative extended convergence continuation tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_xCount</td>
<td>Number of SLP iterations over which to measure static objective (1) convergence</td>
<td>5</td>
</tr>
<tr>
<td>xslp_xLimit</td>
<td>Number of SLP iterations up to which static objective (1) convergence testing starts</td>
<td>100</td>
</tr>
<tr>
<td>xslp_xTol_a</td>
<td>Absolute static objective function (1) tolerance</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_xTol_r</td>
<td>Relative static objective function (1) tolerance</td>
<td>auto</td>
</tr>
</tbody>
</table>

#### 5.42.3.20 NLP Multistart Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_msMaxBoundRange</td>
<td>Defines the maximum range inside which initial points are generated by multistart presets</td>
<td>1000</td>
</tr>
<tr>
<td>xslp_multistartPreset</td>
<td>Enable multistart</td>
<td>0</td>
</tr>
<tr>
<td>xslp_multistart_maxSolves</td>
<td>Maximum number of jobs to create during the multistart search</td>
<td>unlimited</td>
</tr>
<tr>
<td>xslp_multistart_maxTime</td>
<td>Maximum total time to be spent in the multistart search</td>
<td>unlimited</td>
</tr>
<tr>
<td>xslp_multistart_poolsize</td>
<td>Maximum number of problem objects allowed to pool up before synchronization in the deterministic multistart</td>
<td>2</td>
</tr>
<tr>
<td>xslp_multistart_seed</td>
<td>Random seed used for the automatic generation of initial point when loading multistart presets</td>
<td>0</td>
</tr>
<tr>
<td>xslp_multistart_threads</td>
<td>Maximum number of threads to be used in multistart</td>
<td>auto</td>
</tr>
</tbody>
</table>

#### 5.42.3.21 NLP Derivative Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_cdTol_a</td>
<td>Absolute tolerance for deducing constant derivatives</td>
<td>1e-08</td>
</tr>
<tr>
<td>xslp_cdTol_r</td>
<td>Relative tolerance for deducing constant derivatives</td>
<td>1e-08</td>
</tr>
<tr>
<td>xslp_deltaZLimit</td>
<td>Number of SLP iterations during which to apply xslp_delta_z</td>
<td>0</td>
</tr>
<tr>
<td>xslp_delta_a</td>
<td>Absolute perturbation of values for calculating numerical derivatives</td>
<td>0.001</td>
</tr>
<tr>
<td>xslp_delta_r</td>
<td>Relative perturbation of values for calculating numerical derivatives</td>
<td>0.001</td>
</tr>
<tr>
<td>xslp_delta_z</td>
<td>Tolerance used when calculating derivatives</td>
<td>1e-05</td>
</tr>
<tr>
<td>xslp_delta_zero</td>
<td>Absolute zero acceptance tolerance used when calculating derivatives</td>
<td>-1</td>
</tr>
<tr>
<td>xslp_derivatives</td>
<td>Bitmap describing the method of calculating derivatives</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.42.3.22 NLP Log Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_analyze</td>
<td>Bit map activating additional options supporting model / solution path analysis</td>
<td>0</td>
</tr>
<tr>
<td>xslp_analyze_extendedFinalSummary</td>
<td>Include an extended iteration summary</td>
<td>0</td>
</tr>
<tr>
<td>xslp_analyze_infeasibleIteration</td>
<td>Run infeasibility analysis on infeasible iterations</td>
<td>0</td>
</tr>
<tr>
<td>xslp_analyze_saveFile</td>
<td>Create an Xpress SLP save file at every xslp_autosave iterations</td>
<td>0</td>
</tr>
<tr>
<td>xslp_analyze_saveIterBasis</td>
<td>Write the initial basis of the linearizations to disk at every xslp_autosave iterations</td>
<td>0</td>
</tr>
<tr>
<td>xslp_analyze_saveLinearizations</td>
<td>Write the linearizations to disk at every xslp_autosave iterations</td>
<td>0</td>
</tr>
<tr>
<td>xslp_autosave</td>
<td>Frequency with which to save the model</td>
<td>0</td>
</tr>
<tr>
<td>xslp_log</td>
<td>Level of printing during SLP iterations</td>
<td>0</td>
</tr>
<tr>
<td>xslp_slpLog</td>
<td>Frequency with which SLP status is printed</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.42.3.23 MINLP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslp_cutStrategy</td>
<td>Determines which cuts to apply in the MISLP search when the default SLP-in-MIP strategy is used</td>
<td>0</td>
</tr>
<tr>
<td>xslp_heurStrategy</td>
<td>Branch and Bound: MINLP heuristic strategy</td>
<td>auto</td>
</tr>
<tr>
<td>xslp_mipAlgorithm</td>
<td>Bitmap describing the MISLP algorithms to be used</td>
<td>17</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_finalFixSLP</td>
<td>Fix step bounds according to xslp_mipFixStepBounds after MIP solution is found</td>
<td>0</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_finalRelaxSLP</td>
<td>Relax step bounds according to xslp_mipRelaxStepBounds after MIP solution is found</td>
<td>0</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_initialFixSLP</td>
<td>Fix step bounds according to xslp_mipFixStepBounds after initial node</td>
<td>0</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_initialRelaxSLP</td>
<td>Relax step bounds according to xslp_mipRelaxStepBounds after initial node</td>
<td>0</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_initialSLP</td>
<td>Solve initial SLP to convergence</td>
<td>1</td>
</tr>
<tr>
<td>xslp_mipAlgorithm_nodeFixSLP</td>
<td>Fix step bounds according to xslp_mipFixStepBounds at each node</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>xslp_mipAlgorithm_nodeLimitSLP</code></td>
<td>Limit iterations at each node to <code>xslp_mipIterLimit</code></td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipAlgorithm_nodeRelaxSLP</code></td>
<td>Relax step bounds according to <code>xslp_mipRelaxStepBounds</code> at each node</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_mipAlgorithm_slpThenMIP</code></td>
<td>Use MIP on converged SLP solution and then SLP on the resulting MIP solution</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipAlgorithm_withinSLP</code></td>
<td>Use MIP at each SLP iteration instead of SLP at each node</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipCutOffCount</code></td>
<td>Number of SLP iterations to check when considering a node for cutting off</td>
<td>5</td>
</tr>
<tr>
<td><code>xslp_mipCutOffLimit</code></td>
<td>Number of SLP iterations to check when considering a node for cutting off</td>
<td>10</td>
</tr>
<tr>
<td><code>xslp_mipCutOff_a</code></td>
<td>Absolute objective function cutoff for MIP termination</td>
<td>1e-05</td>
</tr>
<tr>
<td><code>xslp_mipCutOff_r</code></td>
<td>Absolute objective function cutoff for MIP termination</td>
<td>1e-05</td>
</tr>
<tr>
<td><code>xslp_mipDefaultAlgorithm</code></td>
<td>Default algorithm to be used during the global search in MISLP</td>
<td>auto</td>
</tr>
<tr>
<td><code>xslp_mipErrorTol_a</code></td>
<td>Absolute penalty error cost tolerance for MIP cut-off</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipErrorTol_r</code></td>
<td>Relative penalty error cost tolerance for MIP cut-off</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipFixStepBounds</code></td>
<td>Bitmap describing the step-bound fixing strategy during MISLP</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipFixStepBounds_coef</code></td>
<td>Fix step bounds on SLP variables appearing in coefficients</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipFixStepBounds_coefOnly</code></td>
<td>Fix step bounds on SLP variables appearing only in coefficients</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipFixStepBounds_structAll</code></td>
<td>Fix step bounds on all structural SLP variables</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipFixStepBounds_structNotCoef</code></td>
<td>Fix step bounds on structural SLP variables which are not in coefficients</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipIterLimit</code></td>
<td>Maximum number of SLP iterations at each node</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipLog</code></td>
<td>Frequency with which MIP status is printed</td>
<td>0</td>
</tr>
<tr>
<td><code>xslp_mipOCount</code></td>
<td>Number of SLP iterations at each node over which to measure objective function variation</td>
<td>5</td>
</tr>
<tr>
<td><code>xslp_mipOTol_a</code></td>
<td>Absolute objective function tolerance for MIP termination</td>
<td>1e-05</td>
</tr>
<tr>
<td><code>xslp_mipOTol_r</code></td>
<td>Relative objective function tolerance for MIP termination</td>
<td>1e-05</td>
</tr>
<tr>
<td><code>xslp_mipRelaxStepBounds</code></td>
<td>Bitmap describing the step-bound relaxation strategy during MISLP</td>
<td>15</td>
</tr>
<tr>
<td><code>xslp_mipRelaxStepBounds_coef</code></td>
<td>Relax step bounds on SLP variables appearing in coefficients</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_mipRelaxStepBounds_coefOnly</code></td>
<td>Relax step bounds on SLP variables appearing only in coefficients</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_mipRelaxStepBounds_structAll</code></td>
<td>Relax step bounds on all structural SLP variables</td>
<td>1</td>
</tr>
<tr>
<td><code>xslp_mipRelaxStepBounds_structNotCoef</code></td>
<td>Relax step bounds on structural SLP variables which are not in coefficients</td>
<td>1</td>
</tr>
</tbody>
</table>
5.42.4 Detailed Descriptions of XPRESS Options

advBasis (boolean): Use advanced basis provided by GAMS

Default: auto

algAfterCrossover (integer): Algorithm to be used for the final clean up step after the crossover

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex.</td>
</tr>
<tr>
<td>3</td>
<td>Primal simplex.</td>
</tr>
<tr>
<td>4</td>
<td>Concurrent.</td>
</tr>
</tbody>
</table>

algAfterNetwork (integer): Algorithm to be used for the clean up step after the network simplex solver

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex.</td>
</tr>
<tr>
<td>3</td>
<td>Primal simplex.</td>
</tr>
</tbody>
</table>

algorithm (string): Choose between simplex and barrier algorithm

This option is used to select the barrier method to solve LPs. By default the barrier method will do a crossover to find a basic solution.

Default: simplex

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>barrier</td>
<td>Use the barrier algorithm</td>
</tr>
<tr>
<td>simplex</td>
<td>Use the simplex algorithm</td>
</tr>
</tbody>
</table>

autoCutting (integer): Automatically decide whether to generate cutting planes at local nodes in the tree

If the cutFreq control is set, no automatic selection will be made and local cutting will be enabled.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled.</td>
</tr>
</tbody>
</table>
autoPerturb (boolean): Simplex: Indicates whether automatic perturbation is performed

If this is set to 1, the problem will be perturbed whenever the simplex method encounters an excessive number of degenerate pivot steps, thus preventing the Optimizer being hindered by degeneracies.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No perturbation performed.</td>
</tr>
<tr>
<td>1</td>
<td>Automatic perturbation is performed.</td>
</tr>
</tbody>
</table>

autoScaling (integer): Whether the Optimizer should automatically select between different scaling algorithms

If the scaling control is set, no automatic scaling will be applied.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Cautious strategy. Non-standard scaling will only be selected if it appears to be clearly superior.</td>
</tr>
<tr>
<td>2</td>
<td>Moderate strategy.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive strategy. Standard scaling will only be selected if it appears to be clearly superior.</td>
</tr>
</tbody>
</table>

backTrack (integer): Branch and Bound: Specifies how to select the next node to work on when a full backtrack is performed

Note: When two nodes are rated the same according to the backTrack selection, a secondary rating is performed using the method set by backtrackTie.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>1</td>
<td>Unused.</td>
</tr>
<tr>
<td>2</td>
<td>Select the node with the best estimated solution.</td>
</tr>
<tr>
<td>3</td>
<td>Select the node with the best bound on the solution.</td>
</tr>
<tr>
<td>4</td>
<td>Select the deepest node in the search tree (equivalent to depth-first search).</td>
</tr>
<tr>
<td>5</td>
<td>Select the highest node in the search tree (equivalent to breadth-first search).</td>
</tr>
<tr>
<td>6</td>
<td>Select the earliest node created.</td>
</tr>
<tr>
<td>7</td>
<td>Select the latest node created.</td>
</tr>
<tr>
<td>8</td>
<td>Select a node randomly.</td>
</tr>
<tr>
<td>9</td>
<td>Select the node whose LP relaxation contains the fewest number of infeasible global entities.</td>
</tr>
<tr>
<td>10</td>
<td>Combination of 2 and 9.</td>
</tr>
<tr>
<td>11</td>
<td>Combination of 2 and 4.</td>
</tr>
<tr>
<td>12</td>
<td>Combination of 3 and 4.</td>
</tr>
</tbody>
</table>
backtrackTie (integer): Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed →

The options are the same as for the backTrack control.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Default selection.</td>
</tr>
<tr>
<td>1</td>
<td>Unused.</td>
</tr>
<tr>
<td>2</td>
<td>Select the node with the best estimated solution.</td>
</tr>
<tr>
<td>3</td>
<td>Select the node with the best bound on the solution.</td>
</tr>
<tr>
<td>4</td>
<td>Select the deepest node in the search tree (equivalent to depth-first search).</td>
</tr>
<tr>
<td>5</td>
<td>Select the highest node in the search tree (equivalent to breadth-first search).</td>
</tr>
<tr>
<td>6</td>
<td>Select the earliest node created.</td>
</tr>
<tr>
<td>7</td>
<td>Select the latest node created.</td>
</tr>
<tr>
<td>8</td>
<td>Select a node randomly.</td>
</tr>
<tr>
<td>9</td>
<td>Select the node whose LP relaxation contains the fewest number of infeasible global entities.</td>
</tr>
<tr>
<td>10</td>
<td>Combination of 2 and 9.</td>
</tr>
<tr>
<td>11</td>
<td>Combination of 2 and 4.</td>
</tr>
<tr>
<td>12</td>
<td>Combination of 3 and 4.</td>
</tr>
</tbody>
</table>

barAlg (integer): Determines which barrier algorithm is to be used to solve the problem →

The automatic setting uses 1 for LP and QP problems and 3 for QCQP problems. Usually the detection of primal or dual infeasibility is more robust with settings 2 or 3, therefore, it is advantageous to use one of these values if the model is presumably infeasible.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Unused.</td>
</tr>
<tr>
<td>1</td>
<td>Use the infeasible-start barrier algorithm.</td>
</tr>
<tr>
<td>2</td>
<td>Use the homogeneous self-dual barrier algorithm.</td>
</tr>
<tr>
<td>3</td>
<td>Start with 2 and optionally switch to 1 during the execution.</td>
</tr>
</tbody>
</table>

barCores (integer): If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm →

If the value is set to the default value (-1), Xpress will automatically detect the number of cores.

The control is provided for cross-hardware reproducibility purposes. The count does not include logical cores created by Hyper-Threading.

Range: \{-1, ..., \infty\}

Default: auto

barCrash (integer): Newton barrier: Determines the type of crash used for the crossover →
During the crash procedure, an initial basis is determined which attempts to speed up the crossover. A good choice at this stage will significantly reduce the number of iterations required to crossover to an optimal solution. The possible values increase proportionally to their time-consumption.

Default: 4

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turns off all crash procedures.</td>
</tr>
<tr>
<td>1–6</td>
<td>Available strategies with 1 being conservative and 6 being aggressive.</td>
</tr>
</tbody>
</table>

**barDualStop** *(real)*: Newton barrier: Convergence parameter, representing the tolerance for dual infeasibilities

If the difference between the constraints and their bounds in the dual problem falls below this tolerance in absolute value, optimization will stop and the current solution will be returned.

Range: \([0, \infty]\)

Default: auto

**barFailIterLimit** *(integer)*: Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Maximum number of consecutive barrier iterations allowed without progress.</td>
</tr>
</tbody>
</table>

**barFreeScale** *(real)*: Defines how the barrier algorithm scales free variables

When using smaller values the barrier algorithm scales free variables more aggressively which can improve performance but may impact numerical stability.

Range: \([0, \infty]\)

Default: \(1e^{-06}\)

**barGapStop** *(real)*: Newton barrier: Convergence parameter, representing the tolerance for the relative duality gap

When the difference between the primal and dual objective function values falls below this tolerance, the Optimizer determines that the optimal solution has been found.

Range: \([0, \infty]\)

Default: auto

**barGapTarget** *(real)*: Newton barrier: Target tolerance for the relative duality gap
The barrier algorithm will keep iterating until either `barGapTarget` is satisfied or until no further improvements are possible. In the latter case, if `barGapStop` is satisfied, it will declare the problem optimal.

When a solution returned by the barrier algorithm has not converged tightly enough for an application, for example if the dual solution is not accurate enough or crossover is taking too long, setter `barGapTarget` to a small value often resolves the problem, without the risk of the solve failing due to a complementarity level not being numerically achievable. Typical suggested values can be between 1-10 and 1-18.

Range: \([0, \infty]\)

Default: auto

**barIndefLimit** (`integer`): Newton Barrier: Limits the number of consecutive indefinite barrier iterations that will be performed

The optimizer will try to minimize (resp. maximize) a QP problem even if the Q matrix is not positive (resp. negative) semi-definite. However, the optimizer may detect that the Q matrix is indefinite and this can result in the optimizer not converging. This control specifies how many indefinite iterations may occur before the optimizer stops and reports that the problem is indefinite. It is usual to specify a value greater than one, and only stop after a series of indefinite matrices, as the problem may be found to be indefinite incorrectly on a few iterations for numerical reasons.

Range: \(\{1, \ldots, \infty\}\)

Default: 15

**barIterLimit** (`integer`): Newton barrier: Maximum number of iterations

While the simplex method usually performs a number of iterations which is proportional to the number of constraints (rows) in a problem, the barrier method standardly finds the optimal solution to a given accuracy after a number of iterations which is independent of the problem size. The penalty is rather that the time for each iteration increases with the size of the problem. `barIterLimit` specifies the maximum number of iterations which will be carried out by the barrier.

Range: \(\{0, \ldots, \infty\}\)

Default: 500

**barKernel** (`real`): Newton barrier: Defines how centrality is weighted in the barrier algorithm

Increasing this parameter may increase the number of iterations, therefore the recommended range is \([1,2]\) and \([-2,-1]\).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\geq+1.0)</td>
<td>Increases the emphasis on centrality when larger value is set.</td>
</tr>
<tr>
<td>(\leq-1.0)</td>
<td>Selects a value adaptively in every iteration from ([+1, -\text{barKernel}]).</td>
</tr>
</tbody>
</table>

**barObjPerturb** (`real`): Defines how the barrier perturbs the objective

The perturbation scale should be set carefully with consideration to the optimality tolerance. The parameter affects only the barrier solve.
Default: \(1e^{-06}\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off objective perturbation.</td>
</tr>
<tr>
<td>(&gt;0)</td>
<td>Let the optimizer decide if the objective is perturbed or not and use the parameter value as the scale of the perturbation.</td>
</tr>
<tr>
<td>(&lt;0)</td>
<td>Always perturb the objective by the absolute value of the parameter.</td>
</tr>
</tbody>
</table>

**barObjScale (real):** Defines how the barrier scales the objective

The scaling performed by the barrier is applied on top of any other scaling in the problem and only affects the barrier solve.

Default: \(\text{auto}\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>Let the optimizer decide.</td>
</tr>
<tr>
<td>0</td>
<td>Scale by geometric mean.</td>
</tr>
<tr>
<td>(\geq0)</td>
<td>Scale such that the largest objective coefficient’s largest element does not exceed this number. In quadratic problems, the quadratic diagonal is used as reference values instead of the linear objective.</td>
</tr>
</tbody>
</table>

**barOrder (integer):** Newton barrier: Controls the Cholesky factorization in the Newton-Barrier

Default: \(\text{auto}\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Choose automatically.</td>
</tr>
<tr>
<td>1</td>
<td>Minimum degree method. This selects diagonal elements with the smallest number of nonzeros in their rows or columns.</td>
</tr>
<tr>
<td>2</td>
<td>Minimum local fill method. This considers the adjacency graph of nonzeros in the matrix and seeks to eliminate nodes that minimize the creation of new edges.</td>
</tr>
<tr>
<td>3</td>
<td>Nested dissection method. This considers the adjacency graph and recursively seeks to separate it into non-adjacent pieces.</td>
</tr>
</tbody>
</table>

**barOrderThreads (integer):** If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton-barrier method

Larger values than \(\text{barCores}\) will be automatically reduced to the value of \(\text{barCores}\).

Range: \(\{0, \ldots, \infty\}\)

Default: \(\text{auto}\)

**barOutput (boolean):** Newton barrier: Level of solution output provided

Output is provided either after each iteration of the algorithm, or else can be turned off completely by this parameter.

Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No output.</td>
</tr>
<tr>
<td>1</td>
<td>At each iteration.</td>
</tr>
</tbody>
</table>

**barPerturb** (*real*): Newton barrier: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties

`barPerturb` controls how much perturbation is allowed during the barrier iterations. By default no perturbation is allowed. Set this parameter with care as larger perturbations may lead to less efficient iterates and the best settings are problem-dependent.

Range: `[0, \infty)`

Default: 0

**barPresolveOps** (*integer*): Newton barrier: Controls the Newton-Barrier specific presolve operations

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use standard presolve.</td>
</tr>
<tr>
<td>1</td>
<td>Extra effort is spent in barrier specific presolve.</td>
</tr>
<tr>
<td>2</td>
<td>Do full matrix eliminations (reduce matrix size).</td>
</tr>
</tbody>
</table>

**barPrimalStop** (*real*): Newton barrier: Convergence parameter, indicating the tolerance for primal infeasibilities

If the difference between the constraints and their bounds in the primal problem falls below this tolerance in absolute value, the Optimizer will terminate and return the current solution.

Range: `[0, \infty]`

Default: `auto`

**barRefIter** (*integer*): Newton barrier: After terminating the barrier algorithm, further refinement steps can be performed

Such refinement steps are especially helpful if the solution is near to the optimum and can improve primal feasibility and decrease the complementarity gap. It is also often advantageous for the crossover algorithm. `barRefIter` specifies the maximum number of such refinement iterations.

Range: `{0, ..., \infty}`

Default: 0

**barRegularize** (*integer*): Determines how the barrier algorithm applies regularization on the KKT system

The parameter is a bit set but value -1 (the default value) is treated in a special way: if the parameter is set to -1 then the solver will automatically select the bits it deems most useful.

Default: -1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Standard regularization is turned on/off.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Reduced regularization is turned on/off. This option reduces the perturbation effect of the standard regularization.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Forces to keep dependent rows in the KKT system.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Forces to preserve degenerate rows in the KKT system.</td>
</tr>
</tbody>
</table>

barRhsScale *(real)*: Defines how the barrier scales the right hand side

The scaling performed by the barrier is applied on top of any other scaling in the problem and only affects the barrier solve.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Let the optimizer decide.</td>
</tr>
<tr>
<td>0</td>
<td>Scale by geometric mean.</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>Scale such that the largest right hand side coefficient's largest element does not exceed this number.</td>
</tr>
</tbody>
</table>

barStart *(integer)*: Newton barrier: Controls the computation of the starting point for the barrier algorithm

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Uses the available solution for warm-start.</td>
</tr>
<tr>
<td>0</td>
<td>Determine automatically.</td>
</tr>
<tr>
<td>1</td>
<td>Uses simple heuristics to compute the starting point based on the magnitudes of the matrix entries.</td>
</tr>
<tr>
<td>2</td>
<td>Uses the pseudoinverse of the constraint matrix to determine primal and dual initial solutions. Less sensitive to scaling and numerically more robust, but in several case less efficient than 1.</td>
</tr>
<tr>
<td>3</td>
<td>Uses the unit starting point for the homogeneous self-dual barrier algorithm.</td>
</tr>
</tbody>
</table>

barStepStop *(real)*: Newton barrier: Convergence parameter, representing the minimal step size

On each iteration of the barrier algorithm, a step is taken along a computed search direction. If that step size is smaller than barStepStop, the Optimizer will terminate and return the current solution.

If the barrier method is making small improvements on barGapStop on later iterations, it may be better to set this value higher, to return a solution after a close approximation to the optimum has been found.

Range: 

Default: 1e-16

barThreads *(integer)*: If set to a positive integer it determines the number of threads implemented to run the Newton-barrier algorithm
If the value is set to the default value (-1), the threads control will determine the number of threads used.

There is a practical upper limit of 50 on the number of parallel threads the optimizer will create.

Range: \{-1, ..., ∞\}

Default: auto

**basisOut (string):** Directs optimizer to output an MPS basis file ←

In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.

Default: none

**bigM (real):** Infeasibility penalty used if the "Big M" method is implemented ←

Range: \([0, ∞]\)

Default: auto

**bigMMethod (boolean):** Simplex: Whether to use the "Big M" method, or the standard phase I (achieving feasibility) and phase II (achieving optimality) ←

In the "Big M" method, the objective coefficients of the variables are considered during the feasibility phase, possibly leading to an initial feasible basis which is closer to optimal. The side-effects involve possible round-off errors due to the presence of the "Big M" factor in the problem.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>For phase I / phase II.</td>
</tr>
<tr>
<td>1</td>
<td>If &quot;Big M&quot; method to be used.</td>
</tr>
</tbody>
</table>

**branchChoice (integer):** Once a global entity has been selected for branching, this control determines which of the branches is solved first ←

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Minimum estimate branch first.</td>
</tr>
<tr>
<td>1</td>
<td>Maximum estimate branch first.</td>
</tr>
<tr>
<td>2</td>
<td>If an incumbent solution exists, solve the branch satisfied by that solution first. Otherwise solve the minimum estimate branch first (option 0).</td>
</tr>
<tr>
<td>3</td>
<td>Solve first the branch that forces the value of the branching variable to move farther away from the value it had at the root node. If the branching entity is not a simple variable, solve the minimum estimate branch first (option 0).</td>
</tr>
</tbody>
</table>

**branchDisj (integer):** Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search ←
Note Split disjunctions are a special form of disjunctions that can be written as \( \sum_j m_j x_j \leq m_0 \lor \sum_j m_j x_j \geq m_0 + 1 \) The split disjunctions created by the optimizer will use a combination of binary or integer variables \( x_j \), with integer coefficients \( m_j \). Split disjunctions for branching will always be created with a default priority value of 400 instead of the default value of 500 for regular entity branches.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection of the strategy.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Cautious strategy. Disjunctive branches will be created only for general integers with a wide range.</td>
</tr>
<tr>
<td>2</td>
<td>Moderate strategy.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive strategy. Disjunctive branches will be created for both binaries and integers.</td>
</tr>
</tbody>
</table>

**branchStructural** (integer): Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search.

Structural branches will often involve branching on more than a single global entity at a time. As a result of a structural branch, a parent node could therefore end up with more than two child nodes, unlike the standard single entity branches. Structural branches will always be created with a default priority value of 400 instead of the default value of 500 for regular entity branches.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled.</td>
</tr>
</tbody>
</table>

**breadthFirst** (integer): Number of nodes to include in the best-first search before switching to the local first search \( \text{nodeSelection} = 4 \)\n
Range: \{0, ..., \( \infty \)\}

Default: 11

**cacheSize** (integer): Newton Barrier: L2 or L3 (see notes) cache size in kB (kilobytes) of the CPU.

On Intel (or compatible) platforms a value of -1 may be used to determine the cache size automatically. If the CPU model is new then the cache size may not be correctly detected by an older release of the software.

1. Specifying the correct cache size can give a significant performance advantage with the Newton barrier algorithm. If the size is unknown, it is better to specify a smaller size. 2. If the size cannot be determined automatically, a default size of 128kB is assumed. 3. Where present, the L3 cache size should be chosen rather than the L2 cache size. 4. For multi-core CPUs, the cache is shared between a subset of the cores. The Optimizer will divide the cacheSize value by the number of cores sharing the cache if >1 Barrier threads are running. 5. Where the CPU is described as having multiple caches ie. 2x6M then the correct cache size to use is 6M not 12M. 6. Examples: Intel Core 2 Duo E6400 (2M Cache, 2.13GHz) cacheSize=2048 Intel Xeon x5570 (8M Cache, 2.93GHz) cacheSize=8192 Intel Core 2 QX6700 (2x4M Cache, 2.93 GHz) cacheSize=4096. 7. If in doubt, please contact Xpress Support for advice.
Range: \{-1, \ldots, \infty\}

Default: auto

**choleskyAlg** *(integer):* Newton barrier: Type of Cholesky factorization used

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Matrix blocking: 0: automatic setting; 1: manual setting.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>If manual selection of matrix blocking: 0: multi-pass; 1: single-pass.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Nonseparable QP relaxation: 0: off; 1: on.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Corrector weight: 0: automatic setting; 1: manual setting.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>If manual selection of corrector weight: 0: off; 1: on.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Refinement: 0: automatic setting; 1: manual setting.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Preconditioned conjugate gradient method (PCGM): 0: PCGM off; 1: PCGM on.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Preconditioned quasi minimal residual (QMR) to refine solution: 0: QMR off; 1: QMR on.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Perform refinement on the augmented system 0: off; 1: on.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Force highest accuracy in refinement 0: off; 1: on.</td>
</tr>
</tbody>
</table>

**choleskyTol** *(real):* Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm

If the absolute value of the pivot element is less than or equal to **choleskyTol**, it merits special treatment in the Cholesky decomposition process.

Range: \([0, \infty]\)

Default: \(1e-15\)

**clamping** *(integer):* Allows for the adjustment of returned solution values such that they are always within bounds

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Equivalent to <code>clamping_primal</code>.</td>
</tr>
<tr>
<td>1</td>
<td>Equivalent to <code>clamping_dual</code>.</td>
</tr>
<tr>
<td>2</td>
<td>Equivalent to <code>clamping_slacks</code>.</td>
</tr>
<tr>
<td>3</td>
<td>Equivalent to <code>clamping_rdj</code>.</td>
</tr>
</tbody>
</table>

**clamping_dual** *(boolean):* Adjust primal slack values to always be within constraint bounds

See also **clamping**.

Default: 0

**clamping_primal** *(boolean):* Adjust primal solution to always be within primal bounds
Slacks if provided will be adjusted accordingly.

See also clamping.

Default: 0

**clamping** *(boolean)*: Adjust reduced costs to always be within dual bounds implied by the primal solution

See also clamping.

Default: 0

**clamping_slacks** *(boolean)*: Adjust dual solution to always be within the dual bounds implied by the slacks

Reduced costs, if provided, will be adjusted accordingly.

See also clamping.

Default: 0

**concurrentThreads** *(integer)*: Determines the number of threads used by the concurrent solver

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of threads to use</td>
</tr>
</tbody>
</table>

**conflictCuts** *(integer)*: Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts

Conflict cuts are in-tree cuts derived from nodes found to be infeasible or cut off, which can be used to cut off other branches of the search tree.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic</td>
</tr>
<tr>
<td>0</td>
<td>Disable conflict cuts.</td>
</tr>
<tr>
<td>1</td>
<td>Cautious application of conflict cuts.</td>
</tr>
<tr>
<td>2</td>
<td>Medium application of conflict cuts.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive application of conflict cuts.</td>
</tr>
</tbody>
</table>

**coresPerCpu** *(integer)*: Used to override the detected value of the number of cores on a CPU

The cache size (either detected or specified via the cacheSize control) used in Barrier methods will be divided by this amount, and this scaled-down value will be the amount of cache allocated to each Barrier thread.

Range: {-1, ..., ∞}

Default: *auto*
coverCuts (integer): Branch and Bound: Number of rounds of lifted cover inequalities at the top node

A lifted cover inequality is an additional constraint that can be particularly effective at reducing the size of the feasible region without removing potential integral solutions. The process of generating these can be carried out a number of times, further reducing the feasible region, albeit incurring a time penalty. There is usually a good payoff from generating these at the top node, since these inequalities then apply to every subsequent node in the tree search.

Range: \{-1, ..., \infty\}

Default: auto

cpuPlatform (integer): Newton Barrier: Selects the AMD or Intel x86 vectorization instruction set that Barrier should run optimized code for

Generic code, SSE2 and AVX optimized code will all result in the same solution path. Using AVX2 might result in a different solution path.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Highest supported [Generic, SSE2, AVX or AVX2].</td>
</tr>
<tr>
<td>-1</td>
<td>Highest supported solve path consistent code [Generic, SSE2 or AVX].</td>
</tr>
<tr>
<td>0</td>
<td>Use generic code compatible with all CPUs.</td>
</tr>
<tr>
<td>1</td>
<td>Use SSE2 optimized code.</td>
</tr>
<tr>
<td>2</td>
<td>Use AVX optimized code.</td>
</tr>
<tr>
<td>3</td>
<td>Use AVX2 optimized code.</td>
</tr>
</tbody>
</table>

cpuTime (integer): How time should be measured when timings are reported in the log and when checking against time limits

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Disable the timer.</td>
</tr>
<tr>
<td>0</td>
<td>Use elapsed time.</td>
</tr>
<tr>
<td>1</td>
<td>Use process time.</td>
</tr>
</tbody>
</table>

crash (integer): Simplex: Determines the type of crash used when the algorithm begins

During the crash procedure, an initial basis is determined which is as close to feasibility and triangularity as possible. A good choice at this stage will significantly reduce the number of iterations required to find an optimal solution. The possible values increase proportionally to their time-consumption.

For primal simplex the non-bit-vector choices are relevant.

For dual simplex the bit-vector choices are relevant.

Default: 2
crossover (integer): Newton barrier: Determines whether the barrier method will cross over to the simplex method when at optimal solution has been found, to provide an end basis and advanced sensitivity analysis information

The full primal and dual solution is available whether or not crossover is used. The crossover must not be disabled if the barrier is used to reoptimize nodes of a MIP. By default crossover will not be performed on QP and MIQP problems.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>No crossover.</td>
</tr>
<tr>
<td>1</td>
<td>Primal crossover first.</td>
</tr>
<tr>
<td>2</td>
<td>Dual crossover first.</td>
</tr>
</tbody>
</table>

crossoverAccuracyTol (real): Newton barrier: Determines how crossover adjusts the default relative pivot tolerance

When re-inversion is necessary, crossover will compare the recalculated working basic solution with the assumed ones just before re-inversion took place. If the error is above this threshold, crossover will adjust the relative pivot tolerance to address the build-up of numerical inaccuracies.

The full primal and dual solution is available whether or not crossover is used. The crossover must not be disabled if the barrier is used to reoptimize nodes of a MIP. By default crossover will not be performed on QP and MIQP problems.

Range: \([0, \infty]\)

Default: 1e-06

crossoverIterLimit (integer): Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates

Range: \(\{0, ..., \infty\}\)

Default: 2147483647

crossoverOps (integer): Newton barrier: Bit vector for adjusting the behavior of the crossover procedure

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Returned solution when the crossover terminates prematurely: 0: Return the last basis from the crossover; 1: Return the barrier solution.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Select the crossover stages to be performed: 0: Perform both crossover stages; 1: Skip second crossover stage.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Set crossover behaviour: 0: Force to perform all pivots; 1: Skip pivots that are numerically less reliable.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Set crossover behaviour: 0: Perform standard crossover; 1: Perform a slower, but numerically more careful crossover.</td>
</tr>
</tbody>
</table>

**crossoverThreads** *(integer)*: Determines the maximum number of threads that parallel crossover is allowed to use

If *crossoverThreads* is set to the default value (-1), the *barThreads* control will determine the number of threads used.

Range: \{-1, ..., \infty\}

Default: *auto*

**cutDepth** *(integer)*: Branch and Bound: Sets the maximum depth in the tree search at which cuts will be generated

Generating cuts can take a lot of time, and is often less important at deeper levels of the tree since tighter bounds on the variables have already reduced the feasible region. A value of 0 signifies that no cuts will be generated.

Does not affect cutting on the root node.

Range: \{-1, ..., \infty\}

Default: *auto*

**cutFactor** *(real)*: Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during global search

The cuts and cut coefficients are limited by *cutFactor* times the number of rows and coefficients in the initial matrix.

A value of 0.0 prevents cuts from being added, and a value of e.g. 1.0 will allow the problem to grow to twice the initial number of rows and coefficients.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Let the optimizer decide on the maximum amount of cuts based on <em>cutStrategy</em>.</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>Multiple of number of rows and coefficients to use.</td>
</tr>
</tbody>
</table>

**cutFreq** *(integer)*: Branch and Bound: Frequency at which cuts are generated in the tree search

If the depth of the node modulo *cutFreq* is zero, then cuts will be generated.

Range: \{-1, ..., \infty\}

Default: *auto*
**cutSelect** *(integer)*: Bit vector providing detailed control of the cuts created for the root node of a global solve

Use `treeCutSelect` to control cuts during the tree search.

The default value is -1 which enables all bits. Any bits not listed in the above table should be left in their default ‘on’ state, since the interpretation of such bits might change in future versions of the optimizer.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <code>cutSelect_clique</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>cutSelect_mir</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>cutSelect_cover</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>cutSelect_mirRowAggregation</code>.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to <code>cutSelect_flowpath</code>.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to <code>cutSelect_implication</code>.</td>
</tr>
<tr>
<td>bit 13 = 8192</td>
<td>Equivalent to <code>cutSelect_liftAndProject</code>.</td>
</tr>
<tr>
<td>bit 14 = 16384</td>
<td>Equivalent to <code>cutSelect_disableCutRows</code>.</td>
</tr>
<tr>
<td>bit 15 = 32768</td>
<td>Equivalent to <code>cutSelect_gubCover</code>.</td>
</tr>
<tr>
<td>bit 16 = 65536</td>
<td>Equivalent to <code>cutSelect_zeroHalf</code>.</td>
</tr>
<tr>
<td>bit 17 = 131072</td>
<td>Equivalent to <code>cutSelect_indicator</code>.</td>
</tr>
</tbody>
</table>

**cutSelect_clique** *(boolean)*: Clique cuts

See also `cutSelect`.

Default: 1

**cutSelect_cover** *(boolean)*: Lifted cover cuts

See also `cutSelect`.

Default: 1

**cutSelect_disableCutRows** *(boolean)*: Disable cutting from cut rows

See also `cutSelect`.

Default: 1

**cutSelect_flowpath** *(boolean)*: Flow path cuts

See also `cutSelect`.

Default: 1

**cutSelect_gubCover** *(boolean)*: Lifted GUB cover cuts

See also `cutSelect`.

Default: 1
**cutSelect_implication** *(boolean):* Implication cuts

See also cutSelect.

Default: 1

**cutSelect_indicator** *(boolean):* Indicator constraint cuts

See also cutSelect.

Default: 1

**cutSelect_liftAndProject** *(boolean):* Turn on automatic Lift-and-Project cutting strategy

See also cutSelect.

Default: 1

**cutSelect_mir** *(boolean):* Mixed Integer Rounding (MIR) cuts

See also cutSelect.

Default: 1

**cutSelect_mirRowAggregation** *(boolean):* Turn on row aggregation for MIR cuts

See also cutSelect.

Default: 1

**cutSelect_zeroHalf** *(boolean):* Zero-half cuts

See also cutSelect.

Default: 1

**cutStrategy** *(integer):* Branch and Bound: Cut strategy

A more aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection of the cut strategy.</td>
</tr>
<tr>
<td>0</td>
<td>No cuts.</td>
</tr>
<tr>
<td>1</td>
<td>Conservative cut strategy.</td>
</tr>
<tr>
<td>2</td>
<td>Moderate cut strategy.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive cut strategy.</td>
</tr>
</tbody>
</table>

**defaultAlg** *(integer):* Selects the algorithm that will be used to solve the LP

Please note that this will affect how the MIP node LP problems are solved during the global search.
Default: auto
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>2</td>
<td>Dual simplex.</td>
</tr>
<tr>
<td>3</td>
<td>Primal simplex.</td>
</tr>
<tr>
<td>4</td>
<td>Newton barrier.</td>
</tr>
</tbody>
</table>

**denseColLimit (integer):** Newton barrier: Controls trigger point for special treatment of dense columns in Cholesky factorization

Columns with more than denseColLimit elements are considered to be dense. Such columns will be handled specially in the Cholesky factorization of this matrix.

Range: \( \{0, ..., \infty\} \)

Default: `auto`

**deterministic (integer):** Branch and Bound: Specifies whether the parallel MIP search should be deterministic

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use non-deterministic parallel MIP.</td>
</tr>
<tr>
<td>1</td>
<td>Use deterministic parallel MIP.</td>
</tr>
<tr>
<td>2</td>
<td>Use deterministic parallel MIP, but allow the concurrent root LP solve to be opportunistic.</td>
</tr>
</tbody>
</table>

**dualGradient (integer):** Simplex: Dual simplex pricing method

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determine automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Devex.</td>
</tr>
<tr>
<td>1</td>
<td>Steepest edge.</td>
</tr>
<tr>
<td>2</td>
<td>Direct steepest edge.</td>
</tr>
<tr>
<td>3</td>
<td>Sparse Devex.</td>
</tr>
</tbody>
</table>

**dualize (integer):** Whether presolve should form the dual of the problem

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determine automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Solve the primal problem.</td>
</tr>
<tr>
<td>1</td>
<td>Solve the dual problem.</td>
</tr>
</tbody>
</table>

**dualizeOps (boolean):** Bit-vector control for adjusting the behavior when a problem is dualized

Default: 1
### dualPerturb (real)
Factor by which the problem will be perturbed prior to optimization by dual simplex

A value of 0.0 results in no perturbation prior to optimization. Note the interconnection to the `autoPerturb` control. If `autoPerturb` is set to 1, the decision whether to perturb or not is left to the Optimizer. When the problem is automatically perturbed in dual simplex, however, the value of `dualPerturb` will be used for perturbation.

Range: $[-\infty, \infty]$

Default: `auto`

### dualStrategy (integer)
Bit-vector control specifies the dual simplex strategy

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Switch to primal when re-optimization goes dual infeasible and numerically unstable.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>When dual intend to switch to primal, stop the solve instead of switching to primal.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Use more aggressive cut-off in MIP search.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Use dual simplex to remove cost perturbations.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Enable more aggressive dual pivoting strategy.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Keep using dual simplex even when it’s numerically unstable.</td>
</tr>
</tbody>
</table>

### dualThreads (integer)
Determines the maximum number of threads that dual simplex is allowed to use

If `dualThreads` is set to the default value (-1), the `threads` control will determine the number of threads used.

When solving a linear MIP, the dual simplex algorithm will use multiple threads only when solving the initial LP relaxation or when reoptimizing between rounds of cuts on the root node. The parallel dual simplex algorithm differs from the sequential dual simplex algorithm and might follow a different solve path. For `dualThreads` > 1 the solve path is independent of the number of threads used, although the practical limit for observing performance benefits is around `dualThreads` = 8.

Range: $\{-1, ..., \infty\}$

Default: `auto`

### eigenvalueTol (real)
Quadratic matrix is considered not to be positive semi-definite, if its smallest eigenvalue is smaller than the negative of this value

Range: $[0, \infty]$

Default: `1e-06`
elimFillin (integer): Amount of fill-in allowed when performing an elimination in presolve

Range: \{0, \ldots, \infty\}
Default: 10

elimTol (real): Markowitz tolerance for the elimination phase of the presolve

Range: [0, \infty]
Default: 0.001

etaTol (real): Tolerance on eta elements

During each iteration, the basis inverse is premultiplied by an elementary matrix, which is the identity except for one column - the eta vector. Elements of eta vectors whose absolute value is smaller than \etaTol are taken to be zero in this step.

Range: [0, \infty]
Default: 1e-13

extraPresolve (integer): No longer has any effect and will be removed from future releases

Use presolveMaxGrow to limit the number of non-zero coefficients in the presolved problem.

Range: \{0, \ldots, \infty\}
Default: 0

feasibilityPump (integer): Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Turned off.</td>
</tr>
<tr>
<td>1</td>
<td>Always try the Feasibility Pump.</td>
</tr>
<tr>
<td>2</td>
<td>Try the Feasibility Pump only if other heuristics have failed to find an integer solution.</td>
</tr>
</tbody>
</table>

feasTol (real): Determines when a solution is treated as feasible

If the amount by which a constraint’s activity violates its right-hand side or ranged bound is less in absolute magnitude than feasTol, then the constraint is treated as satisfied. Similarly, if the amount by which a column violates its bounds is less in absolute magnitude than feasTol, those bounds are also treated as satisfied.

Range: [0, \infty]
Default: 1e-06

feasTolPerturb (real): Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes
The tolerance `feasTol` is always considered as an upper limit for the perturbations, but in some cases smaller value can be more desirable.

Range: $[0, \infty)$

Default: $1e^{-06}$

**feasTolTarget** *(real)*: Target feasibility tolerance for the solution refiner

Zero and negative values are ignored, and the value of `feasTol` is used.

Use very small values like $1e^{-100}$ to state the refinement should continue as long as an improvement is made. Use very large values like $1e+100$ to disable only this aspect of the refiner.

Refining solutions to match the `feasTolTarget` can influence and worsen their objective value in case the previous objective could only be achieved through slight infeasibilities.

Range: $[-\infty, \infty)$

Default: 0

**fixoptfile** *(string)*: name of option file which is read just before solving the fixed problem

**forceParallelDual** *(boolean)*: Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm

By default, when using a single thread, the dual simplex solver will execute a dedicated sequential simplex algorithm.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled. Force the dual simplex solver to use the parallel algorithm.</td>
</tr>
</tbody>
</table>

**genConsAbsTransformation** *(integer)*: Specifies the reformulation method for absolute value general constraints at the beginning of the search

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Use a formulation based on indicator constraints.</td>
</tr>
<tr>
<td>1</td>
<td>Use a formulation based on SOS1-contraints.</td>
</tr>
</tbody>
</table>

**genConsDualReductions** *(boolean)*: Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs

Default: 1
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled. No dual reductions, add columns and rows.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled. Only add necessary columns and rows, drop those implied by the objective sense.</td>
</tr>
</tbody>
</table>

**gomCuts** (*integer*): Branch and Bound: Number of rounds of Gomory or lift-and-project cuts at the top node→

Range: \{-1, ..., \infty\}

Default: *auto*

**heurBeforeLp** (*integer*): Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved→

It is possible that a heuristic will find an optimal integer solution that will result in the LP relaxation solution being cut off. If dedicated heuristic threads are enabled through the **heurThreads** control, then the initial heuristics will be run in parallel with the LP solve, instead of before.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic - let the optimizer decide if heuristics should be run.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled.</td>
</tr>
</tbody>
</table>

**heurDiveIterLimit** (*real*): Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic→

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No iteration limit.</td>
</tr>
<tr>
<td>≥1</td>
<td>Fixed iteration limit.</td>
</tr>
<tr>
<td>&lt;0</td>
<td>Automatic selection of the iteration limit based on the problem size. The absolute value is used as a multiplier on the automatic selection.</td>
</tr>
</tbody>
</table>

**heurDiveRandomize** (*real*): Level of randomization to apply in the diving heuristic→

The diving heuristic uses priority weights on rows and columns to determine the order in which to e.g. round fractional columns, or the direction in which to round them. This control determines by how large a random factor these weights should be changed.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0–1.0</td>
<td>Amount of randomization (0.0=none, 1.0=full)</td>
</tr>
</tbody>
</table>

**heurDiveSoftRounding** (*integer*): Branch and Bound: Enables a more cautious strategy for the diving heuristic, where it tries to push binaries and integer variables to their bounds using the objective, instead
of directly fixing them.

This can be useful when the default diving heuristics fail to find any feasible solutions.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection.</td>
</tr>
<tr>
<td>0</td>
<td>Do not use soft rounding.</td>
</tr>
<tr>
<td>1</td>
<td>Cautious use of the soft rounding strategy.</td>
</tr>
<tr>
<td>2</td>
<td>More aggressive use of the soft rounding strategy.</td>
</tr>
</tbody>
</table>

heurDiveSpeedUp (integer): Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Automatic selection biased towards quality</td>
</tr>
<tr>
<td>-1</td>
<td>Automatic selection biased towards speed.</td>
</tr>
<tr>
<td>0-4</td>
<td>Manual emphasis bias from emphasis on quality (0) to emphasis on speed (4).</td>
</tr>
</tbody>
</table>

heurDiveStrategy (integer): Branch and Bound: Chooses the strategy for the diving heuristic

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection of strategy.</td>
</tr>
<tr>
<td>0</td>
<td>Disables the diving heuristic.</td>
</tr>
<tr>
<td>1-18</td>
<td>Available pre-set strategies for rounding infeasible global entities and reoptimizing during the heuristic dive.</td>
</tr>
</tbody>
</table>

heurEmphasis (integer): Branch and Bound: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap

For problems where the goal is to achieve a small gap but not necessarily solving them to optimality, it is recommended to set heurEmphasis to 1. This setting triggers many additional heuristic calls, aiming for reducing the gap at the beginning of the search, typically at the expense of an increased time for proving optimality.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Optimizer default strategy.</td>
</tr>
<tr>
<td>0</td>
<td>Disables all heuristics.</td>
</tr>
<tr>
<td>1</td>
<td>Focus on reducing the primal-dual gap in the early part of the search.</td>
</tr>
<tr>
<td>2</td>
<td>Extremely aggressive search heuristics.</td>
</tr>
</tbody>
</table>

heurForceSpecialObj (boolean): Branch and Bound: Whether local search heuristics without objective or with an auxiliary objective should always be used, despite the automatic selection of the Optimizer


Deactivated by default.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled. Run special objective heuristics on large problems and even if incumbent exists.</td>
</tr>
</tbody>
</table>

**heurFreq** *(integer)*: Branch and Bound: Frequency at which heuristics are used in the tree search

Heuristics will only be used at a node if the depth of the node is a multiple of heurFreq.

Range: \{-1, ..., \infty\}

Default: -1

**heurSearchEffort** *(real)*: Adjusts the overall level of the local search heuristics

heurSearchEffort is used as a multiplier on the default amount of work the local search heuristics should do. A higher value means the local search heuristics will be run more often and that they are allowed to search larger neighborhoods.

Range: \([0, \infty]\)

Default: 1

**heurSearchFreq** *(integer)*: Branch and Bound: How often the local search heuristic should be run in the tree

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled in the tree.</td>
</tr>
<tr>
<td>(n&gt;0)</td>
<td>Number of nodes between each run.</td>
</tr>
</tbody>
</table>

**heurSearchRootCutFreq** *(integer)*: How frequently to run the local search heuristic during root cutting

This is given as how many cut rounds to perform between runs of the heuristic. Set to zero to avoid applying the heuristic during root cutting. Branch and Bound: This specifies how often the local search heuristic should be run in the tree.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled heuristic during cutting.</td>
</tr>
<tr>
<td>(n&gt;0)</td>
<td>Number of cutting rounds between each run.</td>
</tr>
</tbody>
</table>

**heurSearchRootSelect** *(integer)*: Bit vector control for selecting which local search heuristics to apply on the root node of a global solve
Use `heurSearchTreeSelect` to control local search heuristics during the tree search.

Some of the local search heuristics will benefit from having an existing incumbent solution, but it is not required.

Default: 117

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Local search with a large neighborhood. Potentially slow but is good for finding solutions that differs significantly from the incumbent.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Local search with a small neighborhood centered around a node LP solution.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Local search with a small neighborhood centered around an integer solution. This heuristic will often provide smaller, incremental improvements to an incumbent solution.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Local search with a neighborhood set up through the combination of multiple integer solutions.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Unused</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Local search without an objective function. Called seldom and only when no feasible solution is available.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Local search with an auxiliary objective function. Called seldom and only when no feasible solution is available.</td>
</tr>
</tbody>
</table>

\[\text{heurSearchTreeSelect} \ (\text{integer}): \text{ Bit vector control for selecting which local search heuristics to apply during the tree search of a global solve} \]

Use `heurSearchRootSelect` to control local search heuristics on the root node.

Some of the local search heuristics will benefit from having an existing incumbent solution, but it is not required.

Default: 17

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Local search with a large neighborhood. Potentially slow but is good for finding solutions that differs significantly from the incumbent.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Local search with a small neighborhood centered around a node LP solution.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Local search with a small neighborhood centered around an integer solution. This heuristic will often provide smaller, incremental improvements to an incumbent solution.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Local search with a neighborhood set up through the combination of multiple integer solutions.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Unused</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Local search without an objective function. Called seldom and only when no feasible solution is available.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Local search with an auxiliary objective function. Called seldom and only when no feasible solution is available.</td>
</tr>
</tbody>
</table>

\[\text{heurThreads} \ (\text{integer}): \text{ Branch and Bound: Number of threads to dedicate to running heuristics on the root node} \]

When heuristic threads are enable, the heuristics will be run in parallel with the initial LP solve, if possible, and in parallel with the root cutting.

Default: 0
### historyCosts (integer): Branch and Bound: How to update the pseudo cost for a global entity when a strong branch or a regular branch is applied

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined from the <em>threads</em> control.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled. Heuristics will be run sequentially with the root LP solve and cutting.</td>
</tr>
<tr>
<td>&gt;=1</td>
<td>Number of root threads to dedicate to parallel heuristics.</td>
</tr>
</tbody>
</table>

Default: *auto*

### ifCheckConvexity (boolean): Determines if the convexity of the problem is checked before optimization

Applies to quadratic, mixed integer quadratic and quadratically constrained problems. Checking convexity takes some time, thus for problems that are known to be convex it might be reasonable to switch the checking off.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off convexity checking.</td>
</tr>
<tr>
<td>1</td>
<td>Turn on convexity checking.</td>
</tr>
</tbody>
</table>

### indLinBigM (real): During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough

This control defines the largest BigM for which such a linearized version will be added to the problem in addition to the original constraint. If the BigM is even smaller than *indPreLinBigM*, then the original indicator constraint will additionally be dropped from the problem.

*indLinBigM* should always be at least as large as *indPreLinBigM*. For any value less or equal to *indPreLinBigM*, indicator constraints will never be duplicated and only *indPreLinBigM* is taken into account for linearization.

Range: $[0, \infty)$

Default: 100000

### indPreLinBigM (real): During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough

This control defines the largest BigM for which the original constraint will be replaced by the linearized version. If the BigM is larger than *indPreLinBigM* but smaller than *indLinBigM*,
the linearized row will be added but the original indicator constraint is kept as a numerically stable way to check feasibility.

Replacing an indicator constraint with a BigM row has a side effect on tolerances. In the indicator constraint form, the constraint part is satisfied with feasTol tolerance; while after changing it to BigM form, the constraint also includes the binary indicator variable (with a coefficient up to indPreLinBigM and an integrality tolerance of mipTol), therefore the constraint part of the indicator constraint is satisfied with tolerance feasTol+mipTol×indPreLinBigM.

Range: \([0, \infty]\)

Default: 100

**inputTol** *(real)*: Tolerance on input values elements

If any value is less than or equal to inputTol in absolute value, it is treated as zero. For the internal zero tolerance see matrixTol.

This control needs to be set before inputting the problem, as it has no effect afterwards.

Range: \([0, \infty]\)

Default: 0

**invertFreq** *(integer)*: Simplex: Frequency with which the basis will be inverted

The basis is maintained in a factorized form and on most simplex iterations it is incrementally updated to reflect the step just taken. This is considerably faster than computing the full inverted matrix at each iteration, although after a number of iterations the basis becomes less well-conditioned and it becomes necessary to compute the full inverted matrix. The value of invertFreq specifies the maximum number of iterations between full inversions.

Range: \([-1, ..., \infty]\)

Default: auto

**invertMin** *(integer)*: Simplex: Minimum number of iterations between full inversions of the basis matrix

See the description of invertFreq for details.

Range: \([0, ..., \infty]\)

Default: 3

**ioTimeout** *(integer)*: Maximum number of seconds to wait for an I/O operation before it is cancelled

Range: \([0, ..., \infty]\)

Default: 30

**knitroOptFile** *(string)*: Option file for NLP solver KNITRO

**l1Cache** *(integer)*: Newton barrier: L1 cache size in kB (kilo bytes) of the CPU
On Intel (or compatible) platforms a value of -1 may be used to determine the cache size automatically.

Specifying the correct L1 cache size can give a significant performance advantage with the Newton barrier algorithm. If the size is unknown, it is better to specify a smaller size. If the size cannot be determined automatically on Intel (or compatible) platforms, a default size of 8 kB is assumed.

Range: \{-1, ..., \infty\}

Default: auto

\textbf{lnpBest (integer):} Number of infeasible global entities to create lift-and-project cuts for during each round of Gomory cuts at the top node (see gomCuts) \leftrightarrow

Range: \{0, ..., \infty\}

Default: 50

\textbf{lnpIterLimit (integer):} Number of iterations to perform in improving each lift-and-project cut \leftrightarrow

By setting the number to zero a Gomory cut will be created instead.

Range: \{-1, ..., \infty\}

Default: auto

\textbf{loadMipSol (boolean):} Loads a MIP solution (the initial point) \leftrightarrow

If true, the initial point provided by GAMS will be passed to the optimizer to be treated as an integer feasible point. The optimizer uses the values for the discrete variables only: the level values for the continuous variables are ignored and are calculated by fixing the integer variables and reoptimizing. In some cases, loading an initial MIP solution can improve performance. In addition, there will always be a feasible solution to return.

Default: 0

\textbf{localChoice (integer):} Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree \leftrightarrow

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Never backtrack from the first child, unless it is dropped (infeasible or cut off).</td>
</tr>
<tr>
<td>2</td>
<td>Always solve both child nodes before deciding which child to continue with.</td>
</tr>
<tr>
<td>3</td>
<td>Automatically determined.</td>
</tr>
</tbody>
</table>

\textbf{lpFlags (integer):} Bit-vector control which defines the algorithm for solving an LP problem or the initial LP relaxation of a MIP problem \leftrightarrow

When more than one bit is set, then the LP problem will be solved with the concurrent solver. When this control and \texttt{algorithm} is set at the same time, \texttt{algorithm} will overrule the value of this control.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to lpFlags_dual.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to lpFlags_primal.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to lpFlags_barrier.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to lpFlags_network.</td>
</tr>
</tbody>
</table>

lpFlags_barrier (boolean): Use the barrier method

See also lpFlags.

Default: 0

lpFlags_dual (boolean): Use the dual simplex method

See also lpFlags.

Default: 0

lpFlags_network (boolean): Use the network simplex method

See also lpFlags.

Default: 0

lpFlags_primal (boolean): Use the primal simplex method

See also lpFlags.

Default: 0

lpFolding (integer): Simplex and barrier: Whether to fold an LP problem before solving it

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable LP folding.</td>
</tr>
<tr>
<td>1</td>
<td>Enable LP folding. Attempt to fold all LP problems and MIP initial relaxations.</td>
</tr>
</tbody>
</table>

lpIterLimit (integer): Maximum number of iterations that will be performed by primal simplex or dual simplex before the optimization process terminates

Synonym: iterlim

For MIP problems, this is the maximum total number of iterations over all nodes explored by the Branch and Bound method.

Range: \{0, ..., \infty\}

Default: maxint

lpLog (integer): Simplex: Frequency at which the simplex log is printed

This control only has an effect if lpLogStyle is set to 0.

Default: 100
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Log displayed at the end of the optimization only.</td>
</tr>
<tr>
<td>n&lt;0</td>
<td>Detailed output every n iterations.</td>
</tr>
<tr>
<td>n&gt;0</td>
<td>Summary output every n iterations.</td>
</tr>
</tbody>
</table>

**lpLogDelay** *(real)*: Time interval between two LP log lines

This control only has an effect if **lpLogStyle** is set to 1.

Range: \([0, \infty]\)

Default: 1

**lpLogStyle** *(boolean)*: Simplex: Style of the simplex log

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Simplex log is printed based on simplex iteration count, at a fixed frequency as specified by the <strong>lpLog</strong> control.</td>
</tr>
<tr>
<td>1</td>
<td>Simplex Log is printed based on an estimation of elapsed time, determined by an internal deterministic timer.</td>
</tr>
</tbody>
</table>

**lpRefineIterLimit** *(integer)*: Simplex iteration limit the solution refiner can spend in attempting to increase the accuracy of an LP solution

The solution refiner iteratively attempts to increase the accuracy of the solution until either both **feasTolTarget** and **optimalityTolTarget** is satisfied, or accuracy cannot further be increased, or the effort limit determined by **lpRefineIterLimit** is exhausted.

Range: \([-1, \ldots, \infty]\)

Default: auto

**lpThreads** *(integer)*: Parameter is deprecated and will be removed in a future release

This control is provided for compatibility purposes. Please use **concurrentThreads** instead.

The value of this control is mirrored by the **concurrentThreads** control.

Range: \([-1, \ldots, \infty]\)

Default: -1

**markowitzTol** *(real)*: Markowitz tolerance used for the factorization of the basis matrix

Range: \([0, \infty]\)

Default: 0.01

**matrixTol** *(real)*: Zero tolerance on matrix elements

If the value of a matrix element is less than or equal to **matrixTol** in absolute value, it is treated as zero. The control applies when solving a problem, for an input tolerance see **inputTol**.

Range: \([0, \infty]\)

Default: \(1e-09\)

**maxCutTime** *(integer)*: Maximum amount of time allowed for generation of cutting planes and reoptimization

The limit is checked during generation and no further cuts are added once this limit has been exceeded.

Default: 0
### maxImpliedBound (real)

Presolve: When tighter bounds are calculated during MIP preprocessing, only bounds whose absolute value are smaller than maxImpliedBound will be applied to the problem. For numerically challenging MIP problems, it can sometimes help make the solve more stable by reducing the value of maxImpliedBound to something smaller - e.g. 1.0E+06. It is not recommended to increase this parameter beyond the default of 1.0E+08.

- **Range:** [0, ∞]
- **Default:** 1e+08

### maxLocalBacktrack (integer)

Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node.

If this control is set to k, then the candidate set of nodes for a local backtrack will consist of all active nodes in the subtree rooted at height k above the current node. For example, a setting of 1 will result in only sibling nodes of the current node being considered.

- **Default:** auto

### maxMemoryHard (integer)

Sets the maximum amount of memory in megabytes the optimizer should allocate.

If this limit is exceeded, the solve will terminate. This control is designed to make the optimizer stop in a controlled manner, so that the problem object is valid once termination occurs. The solve state will be set to incomplete. This is different to an out of memory condition in which case the optimizer returns an error. The optimizer may still allocate memory once the limit is exceeded to be able to finish the operations and stop in a controlled manner. When resourceStrategy is enabled, the control also has the same effect as maxMemorySoft and will cause the optimizer to try preserving memory when possible.

- **Range:** {0, ..., ∞}
- **Default:** unlimited

### maxMemorySoft (integer)

When resourceStrategy is enabled, this control sets the maximum amount of memory in megabytes the optimizer targets to allocate.

This may change the solving path, but will not cause the solve to terminate early. To set a hard version of the same, please set maxMemoryHard.

- **Range:** {0, ..., ∞}
- **Default:** unlimited

### maxMipSol (integer)

Branch and Bound: Limit on the number of integer solutions to be found by the Optimizer.
It is possible that during optimization the Optimizer will find the same objective solution from different nodes. However, `maxMipSol` refers to the total number of integer solutions found, and not necessarily the number of distinct solutions.

Setting `maxMipSol=1` can alter the solution path as this will put the emphasis on finding any feasible solution by triggering additional heuristics.

Range: \{0, ..., \infty\}

Default: 0

**maxMipTasks (integer):** Branch-and-Bound: The maximum number of tasks to run in parallel during a MIP solve

The MIP solver will create smaller tasks from individual active nodes or based on local search heuristics. These are tasks that will be executed in parallel by the number of threads set by `mipThreads`.

If `maxMipTasks` is set to a fixed, positive value, the branch-and-bound tree nodes will always be solved in the same deterministic way, independent of the actual number of executing threads implied by `mipThreads`. How a MIP is solved will still depend on the number of threads used for solving the continuous relaxation and therefore on the settings for the controls `barThreads`, `barCores`, `dualThreads` and `concurrentThreads`). To obtain a MIP solve that is completely independent of the number of threads, it is sufficient to set `maxMipTasks`, `forceParallelDual`, `barThreads` and `barCores`. The concurrent LP solver should be avoided in this case.

While you can set this control to large value, the implementation will limit the number of tasks for performance reasons. This limit is currently 32 on 32bit platforms and 256 on 64 bit platforms.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Task limit determined automatically from mipThreads.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Fixed task limit.</td>
</tr>
</tbody>
</table>

**maxNode (integer):** Branch and Bound: Maximum number of nodes that will be explored

Range: \{0, ..., \infty\}

Default: `maxint`

**maxScaleFactor (integer):** Determines the maximum scaling factor that can be applied during scaling

The maximum is provided as an exponent of a power of 2.

Default: 64

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–64</td>
<td>The maximum is provided an exponent of a power of 2.</td>
</tr>
</tbody>
</table>

**maxStallTime (integer):** Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent
Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No stall time limit.</td>
</tr>
<tr>
<td>n&gt;0</td>
<td>If an integer solution has been found, stop MIP search after n seconds without a new incumbent. No effect as long as no solution was found.</td>
</tr>
</tbody>
</table>

**maxTime** (*integer*): Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time

For MIP problems, this is the total time taken to solve all the nodes.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No time limit.</td>
</tr>
<tr>
<td>n&lt;0</td>
<td>Stop in LP or MIP search after n seconds.</td>
</tr>
<tr>
<td>n&gt;0</td>
<td>If an integer solution has been found, stop MIP search after n seconds, otherwise continue until an integer solution is finally found.</td>
</tr>
</tbody>
</table>

**mipAbsCutoff** (*real*): Branch and Bound: If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to *mipAbsCutoff*.

This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.

Range: \([-\infty, \infty]\]

Default: *auto*

**mipAbsStop** (*real*): Branch and Bound: Absolute tolerance determining whether the global search will continue or not

It will terminate if \(|\text{MIPOBJVAL} - \text{BESTBOUND}| \leq \text{mipAbsStop}\) where MIPOBJVAL is the value of the best solution’s objective function, and BESTBOUND is the current best solution bound. For example, to stop the global search when a MIP solution has been found and the Optimizer can guarantee it is within 100 of the optimal solution, set *mipAbsStop* to 100.

Range: \([0, \infty]\]

Default: 0

**mipAddCutoff** (*real*): Branch and Bound: Amount to add to the objective function of the best integer solution found to give the new CURRMIPCUTOFF

Once an integer solution has been found whose objective function is equal to or better than CURRMIPCUTOFF, improvements on this value may not be interesting unless they are better by at least a certain amount. If *mipAddCutoff* is nonzero, it will be added to CURRMIPCUTOFF each time an integer solution is found which is better than this new value. This cuts off sections of the tree whose solutions would not represent substantial improvements in the objective function, saving processor time. The control *mipAbsStop* provides a similar function but works in a different way.

Range: \([-\infty, \infty]\]

Default: 0
mipCleanup (boolean): Clean up the MIP solution (round-fix-solve) to get duals

If nonzero, clean up the integer solution obtained, i.e. round and fix the discrete variables and re-solve as an LP to get some marginal values for the discrete vars.

Default: 1

mipComponents (integer): Determines whether disconnected components in a MIP should be solved as separate MIPs

There can be significant performance benefits from solving disconnected components individual instead of being part of the main branch-and-bound search.

If there are no constraints linking two variables, either directly or indirectly through other variables, they are said to belong to two separate disconnected components. When a problem contains disconnected components of significant size, it can be advantageous to solve each component as a separate MIP. When significant disconnected components are detected in the problem, the solver will switch to a different solve mode where each component is solved separately. This switch will happen after the root node processing has completed and when the solve is about to enter the branch-and-bound search.

Solving disconnected components separately is not compatible with concurrent MIP solves. If concurrent MIP solves has been turned off, disconnected components will be solved as part of the standard branch-and-bound search in each concurrent solve.

Disabling MIP dual reductions through mipDualReductions will also disable the separate solve of disconnected components.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic - let the solver decide.</td>
</tr>
<tr>
<td>0</td>
<td>Disable solving disconnected components separately.</td>
</tr>
<tr>
<td>1</td>
<td>Solve disconnected components separately.</td>
</tr>
</tbody>
</table>

mipConcurrentNodes (integer): Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled

When multiple MIP solves are started, they each run up to the mipConcurrentNodes node limit and only one winning solve is selected for continuing the search with.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic - let the solver decide on a node limit.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of nodes each concurrent solve should complete before a winner is selected.</td>
</tr>
</tbody>
</table>

mipConcurrentSolves (integer): Selects the number of concurrent solves to start for a MIP

Each solve will use a unique random seed for its random number generator, but will otherwise apply the same user controls. The first concurrent solve to complete will have solved the MIP and all the concurrent solves will be terminated at this point. Using concurrent solves can be advantageous when a MIP displays a high level of performance variability.
A node limit is imposed on each concurrent solve, through \texttt{mipConcurrentNodes}. When a concurrent solve reaches this node limit, it will be suspended until all concurrent solves have reached the limit. At this point a winner will be declared, based on which solve made the most progress towards optimality and only the winning solve will continue, using all threading resources. If a concurrent solve completes its MIP search before reaching the node limit, all solves will be stopped.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Enabled. The number of concurrent solves depends on \texttt{mipThreads}.</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>Enabled. The number of concurrent solves to start is given by \texttt{n}.</td>
</tr>
<tr>
<td>0, 1</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

\textbf{mipDualReductions} (\texttt{integer}): Branch and Bound: Limits operations that can reduce the MIP solution space

The \texttt{mipDualReductions} control, when set to a value different from 1 will adjust the values of other controls in order to prevent MIP solver operations that can result in the removal of dominated solutions. For example, dual reductions during preprocessing attempts to remove dominated solutions based on objective arguments, assuming that all constraints are known to the Optimizer. If a problem is detected to have symmetries, the solver might also remove some symmetrical solutions from the search space. In both cases, the set of feasible MIP solutions might be reduced. With default settings, it is only guaranteed that at least one optimal solution remains.

When attempting to collect the \texttt{n}-best solutions, it is recommended to set \texttt{mipDualReductions=2}. This will ensure that the only solutions missed by the enumeration are those that only differ from an existing solution in the values of the continuous variables.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Prevent all dual reductions.</td>
</tr>
<tr>
<td>1</td>
<td>Allow all dual reductions.</td>
</tr>
<tr>
<td>2</td>
<td>Allow dual reductions on continuous variables only.</td>
</tr>
</tbody>
</table>

\textbf{mipFracReduce} (\texttt{integer}): Branch and Bound: Specifies how often the optimizer should run a heuristic to reduce the number of fractional integer variables in the node LP solutions

This heuristic is only applicable to problems that are dual degenerate. These are problems that contain multiple solutions with identical objective function value. The more dual degenerate a problem is, the more likely it will be for this heuristic to have an improving effect.

Default: \texttt{auto}

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Run before and after cutting on the root node.</td>
</tr>
<tr>
<td>2</td>
<td>Run also during root cutting.</td>
</tr>
<tr>
<td>3</td>
<td>Run also during the tree search.</td>
</tr>
</tbody>
</table>
**mipKappaFreq** *(integer)*: Branch and Bound: Specifies how frequently the basis condition number (also known as kappa) should be calculated during the branch-and-bound search ←

The condition number is calculated as the norm of the basis matrix multiplied by the norm of its inverse. This uses the Frobenius norm.

A summary will be printed at the end of the solve, summarizing the collected condition numbers collected:

- Nodes kappa stable: No. of stable sampled nodes (kappa < 10^7)
- Nodes kappa suspicious: No. of suspicious sampled nodes (10^7 ≤ kappa < 10^10)
- Nodes kappa unstable: No. of unstable sampled nodes (10^10 ≤ kappa < 10^13)
- Nodes kappa ill-posed: No. of ill-posed sampled nodes (10^13 ≤ kappa)
- Largest kappa seen: The largest condition number calculated through all sampled nodes.
- Kappa attention level: A measure of how ill-posed the problem is (between 0 and 1).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not calculate condition numbers.</td>
</tr>
<tr>
<td>1</td>
<td>Calculate condition numbers on every node, including after each round of root cutting.</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>Calculate a condition number once per node of every n’th level of the branch-and-bound tree.</td>
</tr>
</tbody>
</table>

**mipLog** *(integer)*: Global print control ←

Default: -100

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No printout in global.</td>
</tr>
<tr>
<td>1</td>
<td>Only print out summary statement at the end.</td>
</tr>
<tr>
<td>2</td>
<td>Print out detailed log at all solutions found.</td>
</tr>
<tr>
<td>3</td>
<td>Print out detailed log at each node.</td>
</tr>
<tr>
<td>-n</td>
<td>Print out summary log at each n th node.</td>
</tr>
</tbody>
</table>

**mipPresolve** *(integer)*: Branch and Bound: Type of integer processing to be performed ←

If set to 0, no processing will be performed.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to mipPresolve_reducedCostFixing.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to mipPresolve_logicPreprocessing.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>[Unused] This bit is no longer used to control probing. Refer to the integer control preProbing for setting probing level during presolve.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to mipPresolve_allowChangeBounds.</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to <code>mipPresolve_dualReductions</code>.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <code>mipPresolve_globalCoefTightening</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>mipPresolve_objBasedReductions</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>mipPresolve_allowTreeRestart</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>mipPresolve_symmetryReductions</code>.</td>
</tr>
</tbody>
</table>

`mipPresolve_allowChangeBounds` (boolean): If node preprocessing is allowed to change bounds on continuous columns ⬤

See also `mipPresolve`.

Default: 1

`mipPresolve_allowTreeRestart` (boolean): Allow the branch-and-bound tree search to be restarted if it appears to be advantageous ⬤

See also `mipPresolve`.

Default: 1

`mipPresolve_dualReductions` (boolean): Dual reductions will be performed at each node ⬤

See also `mipPresolve`.

Default: 1

`mipPresolve_globalCoefTightening` (boolean): Allow global (non-bound) tightening of the problem during the tree search ⬤

See also `mipPresolve`.

Default: 1

`mipPresolve_logicPreprocessing` (boolean): Primal reductions will be performed at each node ⬤

Uses constraints of the node to tighten the range of variables, often resulting in fixing their values. This greatly simplifies the problem and may even determine optimality or infeasibility of the node.

See also `mipPresolve`.

Default: 1

`mipPresolve_objBasedReductions` (boolean): Objective function will be used to find reductions at each node ⬤

See also `mipPresolve`.

Default: 1

`mipPresolve_reducedCostFixing` (boolean): Reduced cost fixing will be performed at each node ⬤
This can simplify the node before it is solved, by deducing that certain variables’ values can be fixed based on additional bounds imposed on other variables at this node.

See also mipPresolve.

Default: 1

mipPresolve_symmetryReductions (boolean): Allow that symmetry is used to presolve the node problem

See also mipPresolve.

Default: 1

mipRampUp (integer): Controls the strategy used by the parallel MIP solver during the ramp-up phase of a branch-and-bound tree search

The branch-and-bound tree search starts from the single root node, and only through branching on this root node and the resulting child nodes, are enough active nodes created to produce sufficient tasks to keep all MIP workers busy. This is referred to as the ramp-up phase of a parallel MIP. In a typical MIP solve, the solutions found during the initial dives will typically provide a significant improvement over the root heuristic solutions. It can therefore be advantageous to let these initial dives run as fast as possible, by limiting resource contention. This can be accomplished by restricting the number of parallel tasks and thereby reducing the memory bus contention. The mipRampUp control can be used to turn this initial task restriction of a parallel MIP solve on or off.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>No special treatment during the ramp-up phase. Always run with the maximal number of tasks.</td>
</tr>
<tr>
<td>1</td>
<td>Limit the number of tasks until the initial dives have completed.</td>
</tr>
</tbody>
</table>

mipRefineIterLimit (integer): Defines an effort limit expressed as simplex iterations for the MIP solution refiner

The limit is per reoptimizations in the MIP refiner.

Range: \{-1, ..., ∞\}

Default: auto

mipRelCutoff (real): Branch and Bound: Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of CURRMIPCUTOFF

The effect is to cut off the search in parts of the tree whose best possible objective function would not be substantially better than the current solution. The control mipRelStop provides a similar functionality but works in a different way.

Range: [0, ∞]

Default: 0
**mipRelStop** *(real):* Branch and Bound: Determines when the global search will terminate

Global search will stop if: \[ | \text{MIPOBJVAL} - \text{BESTBOUND} | \leq \text{mipRelStop} \times \max(| \text{BESTBOUND}|, | \text{MIPOBJVAL}|) \]
where MIPOBJVAL is the value of the best solution’s objective function and BESTBOUND is the current best solution bound. For example, to stop the global search when a MIP solution has been found and the Optimizer can guarantee it is within 5% of the optimal solution, set `mipRelStop` to 0.05.

This control is a stopping criteria only and different values of the control will not affect the solution path before termination. Unlike other stopping criteria, like time and node count, termination on `mipRelStop` will cause the final solution to be declared optimal and the problem to be returned to its original state.

Tolerances, such as `mipRelCutoff` and `mipAbsCutoff`, determine how much the objective value of a new MIP solution has to differ from the incumbent for it to be accepted. These controls therefore also influence the final gap at the end of a MIP solve.

Range: \([0, \infty]\)

Default: 0.0001

**mipRestart** *(integer):* Branch and Bound: Controls strategy for in-tree restarts

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically (XPRS_MIPRESTART_DEFAULT).</td>
</tr>
<tr>
<td>0</td>
<td>Disable in-tree restarts (XPRS_MIPRESTART_OFF).</td>
</tr>
<tr>
<td>1</td>
<td>Allow in-tree restarts at normal aggressiveness (XPRS_MIPRESTART_MODERATE).</td>
</tr>
<tr>
<td>2</td>
<td>Allow in-tree restarts at higher aggressiveness (more likely to trigger a restart) (XPRS_MIPRESTART_AGGRESSIVE).</td>
</tr>
</tbody>
</table>

**mipRestartFactor** *(real):* Branch and Bound: Fine tune initial conditions to trigger an in-tree restart

Use a value > 1 to increase the aggressiveness with which the Optimizer restarts. Use a value < 1 to relax the aggressiveness with which the Optimizer restarts. Note that this control does not affect the initial condition on the gap, which must be set separately.

Range: \([0, \infty]\)

Default: 1

**mipRestartGapThreshold** *(real):* Branch and Bound: Initial gap threshold to delay in-tree restart

The restart is delayed initially if the gap, given as a fraction between 0 and 1, is below this threshold. The optimizer adjusts the threshold every time a restart is delayed. Note that there are other criteria that can delay or prevent a restart.

Range: \([0, 1]\)

Default: 0.02

**mipstopexpr** *(string):* Stopping expression for branch and bound
If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: resusd, nodusd, object, objval. Supported operators are: +, -, *, /, ^, !, ==, <, <=, >, >=, &&, ||, (, ), abs, ceil, exp, floor, log, log10, pow, sqrt. Example:

```plaintext
nodusd >= 1000 && abs(objest - objval) / abs(objval) < 0.1
```

**mipThreads (integer):** If set to a positive integer it determines the number of threads implemented to run the parallel MIP code.

If `mipThreads` is set to the default value (-1), the `threads` control will determine the number of threads used.

Range: `{−1, ..., ∞}`

Default: `auto`

**mipTol (real):** Branch and Bound: Tolerance within which a decision variable’s value is considered to be integral.

Range: `[0, ∞]`

Default: `5e-06`

**mipTolTarget (real):** Target `mipTol` value used by the automatic MIP solution refiner as defined by `refineOps`.

Negative and zero values are ignored.

Refining solutions to match the `mipTolTarget` can influence and worsen their objective value in case the previous objective could only be achieved through slight infeasibilities.

Range: `[-∞, ∞]`

Default: `0`

**mipTrace (string):** Name of MIP trace file

A miptrace file with the specified name will be created. This file records the best integer and best bound values every `mipTraceNode` nodes and at `mipTraceTime`-second intervals.

Default: `none`

**mipTraceNode (integer):** Node interval between MIP trace file entries.

Default: `100`

**mipTraceTime (real):** Time interval, in seconds, between MIP trace file entries.

Default: `5`

**miqcpAlg (integer):** Determines which algorithm is to be used to solve mixed integer quadratic constrained and mixed integer second order cone problems.

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Use the barrier algorithm in the branch and bound algorithm.</td>
</tr>
<tr>
<td>1</td>
<td>Use outer approximations in the branch and bound algorithm.</td>
</tr>
</tbody>
</table>
mpsNameLength (integer): Maximum length of MPS names in characters

Internally it is rounded up to the smallest multiple of 8. MPS names are right padded with blanks. Maximum value is 64.

Range: \( \{0, \ldots, \infty\} \)

Default: 0

mpsOutputFile (string): Name of MPS output file

If specified XPRESS-MP will generate an MPS file corresponding to the GAMS model: the argument is the file name to be used. You can prefix the file name with an absolute or relative path.

Default: none

netCuts (integer): Determines the addition of multi-commodity network cuts to a problem

The parameter is defined as a bit string, and values 1, 2, 4 can be summed up if the user wants more classes of cuts to be added.

If the user wants to add both cut-set inequalities and lifted flow-cover inequalities but not node cut-set inequalities, the value of the control should be set to \(1+4=5\).

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Do not add these cuts.</td>
</tr>
<tr>
<td>1</td>
<td>Add cut-set inequalities.</td>
</tr>
<tr>
<td>2</td>
<td>Add node cut-set inequalities, i.e., cut-set inequalities that are based on a network cut defined on a single network node.</td>
</tr>
<tr>
<td>4</td>
<td>Add lifted flow-cover inequalities.</td>
</tr>
</tbody>
</table>

netStallLimit (integer): Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex, depending on \(\text{algAfterNetwork}\)

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>Automatically determined limit</td>
</tr>
<tr>
<td>0</td>
<td>No limit.</td>
</tr>
<tr>
<td>(n&gt;0)</td>
<td>Limit to (n) network simplex iterations.</td>
</tr>
</tbody>
</table>

nodeProbingEffort (real): Adjusts the overall level of node probing

\(\text{nodeProbingEffort}\) is used as a multiplier on the default amount of work node probing should do. Setting the control to zero disables node probing.

Range: \([0, \infty]\)

Default: 1
**nodeSelection** (*integer*): Branch and Bound: Determines which nodes will be considered for solution once the current node has been solved ⬝

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Local first: Choose between descendant and sibling nodes if available; choose from all outstanding nodes otherwise.</td>
</tr>
<tr>
<td>2</td>
<td>Best first: Choose from all outstanding nodes.</td>
</tr>
<tr>
<td>3</td>
<td>Local depth first: Choose between descendant and sibling nodes if available; choose from the deepest nodes otherwise.</td>
</tr>
<tr>
<td>4</td>
<td>Best first, then local first: Best first is used for the first <code>breadthFirst</code> nodes, after which local first is used.</td>
</tr>
<tr>
<td>5</td>
<td>Pure depth first: Choose from the deepest outstanding nodes.</td>
</tr>
</tbody>
</table>

**numericalEmphasis** (*integer*): How much emphasis to place on numerical stability instead of solve speed ⬝

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic. The emphasis might be influenced by the setting of other controls.</td>
</tr>
<tr>
<td>0</td>
<td>Emphasize speed.</td>
</tr>
<tr>
<td>1</td>
<td>Mild emphasis on numerical stability.</td>
</tr>
<tr>
<td>2</td>
<td>Medium emphasis on numerical stability.</td>
</tr>
<tr>
<td>3</td>
<td>Strong emphasis on numerical stability.</td>
</tr>
</tbody>
</table>

**objGoodEnough** (*real*): Stop once an objective this good is found ⬝

Default: `none`

**objScaleFactor** (*integer*): Custom global objective scaling factor, expressed as a power of 2 ⬝

When set, it overwrites the automatic global objective scaling factor. A value of 0 means no objective scaling. This control is applied for the full solve, and is independent of any extra scaling that may occur specifically for the barrier or simplex solvers. As it is a power of 2, to scale by 16, set the value of the control to 4.

Range: `{0, ..., ∞}`

Default: 0

**optimalityTol** (*real*): Simplex: Zero tolerance for reduced costs ⬝

On each iteration, the simplex method searches for a variable to enter the basis which has a negative reduced cost. The candidates are only those variables which have reduced costs less than the negative value of `optimalityTol`.

Range: `[0, ∞]`

Default: `1e-06`

**optimalityTolTarget** (*real*): Target optimality tolerance for the solution refiner ⬝
Zero and negative values are ignored, and the value of `optimalityTol` is used.

Use very small values like 1e-100 to state the refinement should continue as long as an improvement is made. Use very large values like 1e+100 to disable only this aspect of the refiner.

Refining solutions to match the `optimalityTolTarget` can influence and increase their infeasibility in case the previous feasibility could only be achieved through slight dual violations.

Range: \([-\infty, \infty]\]

Default: 0

**outputControls** *(boolean)*: Toggles the printing of all control settings at the beginning of the search

This includes the printing of controls that have been explicitly assigned to their default value. All unset controls are omitted as they keep their default value.

Setting `outputControls` to 0 has no effect on the function `XPRSdumpcontrols`

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off printing of user-specified control settings.</td>
</tr>
<tr>
<td>1</td>
<td>Print controls.</td>
</tr>
</tbody>
</table>

**outputLog** *(integer)*: Controls the level of output produced by the Optimizer during optimization

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn all output off.</td>
</tr>
<tr>
<td>1</td>
<td>Print all messages.</td>
</tr>
<tr>
<td>3</td>
<td>Print error and warning messages.</td>
</tr>
<tr>
<td>4</td>
<td>Print error messages only.</td>
</tr>
</tbody>
</table>

**outputTol** *(real)*: Zero tolerance on print values

Range: \([0, \infty]\]

Default: 1e-05

**penalty** *(real)*: Minimum absolute penalty variable coefficient

Range: \([0, \infty]\]

Default: `auto`

**pivotTol** *(real)*: Simplex: Zero tolerance for matrix elements

On each iteration, the simplex method seeks a nonzero matrix element to pivot on. Any element with absolute value less than `pivotTol` is treated as zero for this purpose.

Range: \([0, \infty]\]

Default: 1e-09
**ppFactor (real):** Partial pricing candidate list sizing parameter $\leftrightarrow$

Range: $[0, \infty)$

Default: 1

**preAnalyticCenter (integer):** Determines if analytic centers should be computed and used for variable fixing and the generation of alternative reduced costs (-1: Auto 0: Off, 1: Fixing, 2: Redcost, 3: Both) $\leftrightarrow$

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable analytic center presolving.</td>
</tr>
<tr>
<td>1</td>
<td>Use analytic center for variable fixing only.</td>
</tr>
<tr>
<td>2</td>
<td>Use analytic center for reduced cost computation only.</td>
</tr>
<tr>
<td>3</td>
<td>Use analytic centers for both, variable fixing and reduced cost computation.</td>
</tr>
</tbody>
</table>

**preBasisRed (integer):** Determines if a lattice basis reduction algorithm should be attempted as part of presolve $\leftrightarrow$

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable basis reduction.</td>
</tr>
<tr>
<td>1</td>
<td>Enable basis reduction.</td>
</tr>
</tbody>
</table>

**preBndRedCone (integer):** Determines if second order cone constraints should be used for inferring bound reductions on variables when solving a MIP $\leftrightarrow$

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable bound reductions from second order cone constraints.</td>
</tr>
<tr>
<td>1</td>
<td>Enable bound reductions from second order cone constraints.</td>
</tr>
</tbody>
</table>

**preBndRedQuad (integer):** Determines if convex quadratic contraints should be used for inferring bound reductions on variables when solving a MIP $\leftrightarrow$

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable bound reductions from quadratic constraints.</td>
</tr>
<tr>
<td>1</td>
<td>Enable bound reductions from quadratic constraints.</td>
</tr>
</tbody>
</table>

**preCliqueStrategy (integer):** Determines how much effort to spend on clique covers in presolve $\leftrightarrow$

Range: $\{-1, ..., \infty\}$
**preCoefElim** (integer): Presolve: Specifies whether the optimizer should attempt to recombine constraints in order to reduce the number of non zero coefficients when presolving a mixed integer problem.

Default: 2

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Remove as many coefficients as possible.</td>
</tr>
<tr>
<td>2</td>
<td>Cautious eliminations. Will not perform a reduction if it might destroy problem structure useful to e.g. heuristics or cutting.</td>
</tr>
</tbody>
</table>

**preComponents** (integer): Presolve: Determines whether small independent components should be detected and solved as individual subproblems during root node processing.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disable detection of independent components.</td>
</tr>
<tr>
<td>1</td>
<td>Enable detection of independent components.</td>
</tr>
</tbody>
</table>

**preComponentsEffort** (real): Presolve: Adjusts the overall effort for the independent component presolver.

This control affects working limits for the subproblem solving as well as thresholds when it is called. Increase to put more emphasis on component presolving.

Range: \([0, \infty)\]

Default: 1

**preConeDecomp** (integer): Presolve: Decompose regular and rotated cones with more than two elements and apply Outer Approximation on the resulting components.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disable cone decomposition.</td>
</tr>
<tr>
<td>1</td>
<td>Enable cone decomposition by replacing large cones with small ones in the presolved problem.</td>
</tr>
<tr>
<td>2</td>
<td>Similar to 1, plus decomposition is enabled even if the cone variable is fixed.</td>
</tr>
<tr>
<td>3</td>
<td>Cones are decomposed within the Outer Approximation domain only, i.e., the problem maintains the original cones.</td>
</tr>
</tbody>
</table>

**preConvertSeparable** (integer): Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints.

This control is only used in MIQPs and MIQCQPs, and has no effect when used on continuous quadratic problems.
Default: auto
### preDomCol (integer): Presolve
Determine the level of dominated column removal reductions to perform when presolving a mixed integer problem

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disable reformulation.</td>
</tr>
<tr>
<td>1</td>
<td>Enable reformulation to diagonal quadratic constraints.</td>
</tr>
<tr>
<td>2</td>
<td>Similar to 1, plus reduction to second-order cones.</td>
</tr>
<tr>
<td>3</td>
<td>Similar to 2, plus the objective function is converted to a constraint and treated as a quadratic constraint.</td>
</tr>
</tbody>
</table>

Only binary columns will be checked.

Default: auto

### preDomRow (integer): Presolve
Determine the level of dominated row removal reductions to perform when presolving a problem

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Cautious strategy.</td>
</tr>
<tr>
<td>2</td>
<td>All candidate binaries will be checked for domination.</td>
</tr>
<tr>
<td>3</td>
<td>Aggressive strategy. All candidate row combinations will be considered.</td>
</tr>
</tbody>
</table>

### preDupRow (integer): Presolve
Determine the type of duplicate rows to look for and eliminate when presolving a problem

Duplicate rows can also be disabled by clearing the corresponding bit of the presolveOps integer control.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Do not eliminate duplicate rows.</td>
</tr>
<tr>
<td>1</td>
<td>Eliminate only rows that are identical in all variables.</td>
</tr>
<tr>
<td>2</td>
<td>Same as option 1 plus eliminate duplicate rows with simple penalty variable expressions. (MIP only).</td>
</tr>
<tr>
<td>3</td>
<td>Same as option 2 plus eliminate duplicate rows with more complex penalty variable expressions. (MIP only).</td>
</tr>
</tbody>
</table>
**preElimQuad** *(integer)*: Presolve: Allows for elimination of quadratic variables via doubleton rows

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Do not eliminate duplicate rows.</td>
</tr>
<tr>
<td>1</td>
<td>Eliminate at least one quadratic variable for each doubleton row.</td>
</tr>
</tbody>
</table>

**preFolding** *(integer)*: Presolve: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Enabled.</td>
</tr>
</tbody>
</table>

**preImplications** *(integer)*: Presolve: Determines whether to use implication structures to remove redundant rows

If implication sequences are detected, they might also be used in probing.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Do not use implications for sparsification.</td>
</tr>
<tr>
<td>1</td>
<td>Use implications to remove redundant rows.</td>
</tr>
</tbody>
</table>

**preLinDep** *(integer)*: Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Do not check for linearly dependent equality constraints.</td>
</tr>
<tr>
<td>1</td>
<td>Check for and remove linearly dependent equality constraints.</td>
</tr>
</tbody>
</table>

**preObjCutDetect** *(boolean)*: Presolve: Determines whether to check for constraints that are parallel or near parallel to a linear objective function, and which can safely be removed

This reduction applies to MIPs only.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable check and reductions.</td>
</tr>
<tr>
<td>1</td>
<td>Enable check and reductions.</td>
</tr>
</tbody>
</table>
**preProbing** (*integer*): Presolve: Amount of probing to perform on binary variables during presolve

This is done by fixing a binary to each of its values in turn and analyzing the implications.

Default: *auto*

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Let the optimizer decide on the amount of probing.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Light probing — only few implications will be examined.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Full probing — all implications for all binaries will be examined.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Full probing and repeat as long as the problem is significantly reduced.</td>
<td></td>
</tr>
</tbody>
</table>

**presolve** (*integer*): Determines whether presolving should be performed prior to starting the main algorithm

Presolve attempts to simplify the problem by detecting and removing redundant constraints, tightening variable bounds, etc. In some cases, infeasibility may even be determined at this stage, or the optimal solution found.

Memory for presolve is dynamically resized. If the Optimizer runs out of memory for presolve, an error message (245) is produced. Presolve settings 2 and 3 can sometimes make the barrier solves more efficient.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Presolve applied, but a problem will not be declared infeasible if primal infeasibilities are detected. The problem will be solved by the LP optimization algorithm, returning an infeasible solution, which can sometimes be helpful.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Presolve not applied.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Presolve applied.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Presolve applied, but redundant bounds are not removed. This can sometimes increase the efficiency of the barrier algorithm.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Presolve is applied, and bounds detected to be redundant are always removed.</td>
<td></td>
</tr>
</tbody>
</table>

**presolveMaxGrow** (*real*): Limit on how much the number of non-zero coefficients is allowed to grow during presolve, specified as a ratio of the number of non-zero coefficients in the original problem

Range: \([0, \infty]\)

Default: 0.1

**presolveOps** (*integer*): Specifies the operations which are performed during the presolve

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 511

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to <code>presolveOps_singletonColRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to <code>presolveOps_singletonRowRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to <code>presolveOps_forcingRowRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to <code>presolveOps_dualReductions</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to <code>presolveOps_redundantRowRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <code>presolveOps_duplicateColRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>presolveOps_duplicateRowRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>presolveOps_strongDualReductions</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>presolveOps_variableEliminations</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to <code>presolveOps_nolpReductions</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to <code>presolveOps_noGlobalDomainChange</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to <code>presolveOps_noAdvIpReductions</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to <code>presolveOps_noIntVarEliminations</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 14 = 16384</td>
<td>Equivalent to <code>presolveOps_linDependRowRemoval</code>.</td>
<td></td>
</tr>
<tr>
<td>bit 15 = 32768</td>
<td>Equivalent to <code>presolveOps_noIntVarAndSosDetect</code>.</td>
<td></td>
</tr>
</tbody>
</table>

`presolveOps_dualReductions (boolean)`: Dual reductions ←

See also `presolveOps`.

Default: 1

`presolveOps_duplicateColRemoval (boolean)`: Duplicate column removal ←

See also `presolveOps`.

Default: 1

`presolveOps_duplicateRowRemoval (boolean)`: Duplicate row removal ←

See also `presolveOps`.

Default: 1

`presolveOps_forcingRowRemoval (boolean)`: Forcing row removal ←

See also `presolveOps`.

Default: 1

`presolveOps_linDependRowRemoval (boolean)`: Linearly dependant row removal ←

See also `presolveOps`.

Default: 0

`presolveOps_noAdvIpReductions (boolean)`: No advanced IP reductions ←

See also `presolveOps`.

Default: 0

`presolveOps_noGlobalDomainChange (boolean)`: No semi-continuous variable detection ←

See also `presolveOps`.

Default: 0
presolveOps_noIntVarAndSosDetect (boolean): No integer variable and SOS detection

See also presolveOps.

Default: 0

presolveOps_noIntVarEliminations (boolean): No eliminations on integers

See also presolveOps.

Default: 0

presolveOps_noIpReductions (boolean): No IP reductions

See also presolveOps.

Default: 0

presolveOps_redundantRowRemoval (boolean): Redundant row removal

See also presolveOps.

Default: 1

presolveOps_singletonColRemoval (boolean): Singleton column removal

See also presolveOps.

Default: 1

presolveOps_singletonRowRemoval (boolean): Singleton row removal

See also presolveOps.

Default: 1

presolveOps_strongDualReductions (boolean): Strong dual reductions

See also presolveOps.

Default: 1

presolveOps_variableEliminations (boolean): Variable eliminations

See also presolveOps.

Default: 1

presolvePasses (integer): Number of reduction rounds to be performed in presolve

Range: \{0, ..., \infty\}

Default: 1

pricingAlg (integer): Simplex: Determines the primal simplex pricing method

It is used to select which variable enters the basis on each iteration. In general Devex pricing requires more time on each iteration, but may reduce the total number of iterations, whereas partial pricing saves time on each iteration, but may result in more iterations.

Default: auto
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Partial pricing.</td>
</tr>
<tr>
<td>0</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>1</td>
<td>Devex pricing.</td>
</tr>
<tr>
<td>2</td>
<td>Steepest edge.</td>
</tr>
<tr>
<td>3</td>
<td>Steepest edge with unit initial weights.</td>
</tr>
</tbody>
</table>

\textbf{primalPerturb} \textit{(real):} Factor by which the problem will be perturbed prior to optimization by primal simplex ⇐

A value of 0.0 results in no perturbation prior to optimization. Note the interconnection to the \texttt{autoPerturb} control. If \texttt{autoPerturb} is set to 1, the decision whether to perturb or not is left to the Optimizer. When the problem is automatically perturbed in primal simplex, however, the value of \texttt{primalPerturb} will be used for perturbation.

Range: \([-\infty, \infty]\)

Default: \texttt{auto}

\textbf{primalUnshift} \textit{(boolean):} Determines whether primal is allowed to call dual to unshift ⇐

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Allow the dual algorithm to be used to unshift.</td>
</tr>
<tr>
<td>1</td>
<td>Don’t allow the dual algorithm to be used to unshift.</td>
</tr>
</tbody>
</table>

\textbf{pseudoCost} \textit{(real):} Branch and Bound: Default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search ⇐

A pseudo cost is associated with each integer decision variable and is an estimate of the amount by which the objective function will be worse if that variable is forced to an integral value.

Range: \([0, \infty]\)

Default: 0.01

\textbf{qcCuts} \textit{(integer):} Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation ⇐

This control only has an effect for problems with quadratic or second order cone constraints, and only if outer approximation has not been disabled by setting \texttt{miqcpAlg} to 0.

Range: \([-1, ..., \infty]\)

Default: \texttt{auto}

\textbf{qcRootAlg} \textit{(integer):} Determines which algorithm is to be used to solve the root of a mixed integer quadratic constrained or mixed integer second order cone problem, when outer approximation is used ⇐

This control only has an effect if \texttt{miqcpAlg} is set to 1.

Default: \texttt{auto}
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Use the barrier algorithm.</td>
</tr>
<tr>
<td>1</td>
<td>Use the dual simplex on a relaxation of the problem constructed using outer approximation.</td>
</tr>
</tbody>
</table>

**qextractalg (integer):** quadratic extraction algorithm in GAMS interface

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic</td>
</tr>
<tr>
<td>1</td>
<td>ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.</td>
</tr>
<tr>
<td>2</td>
<td>DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.</td>
</tr>
</tbody>
</table>

**qSimplexOps (integer):** Controls the behavior of the quadratic simplex solvers

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Force traditional primal first phase.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Force BigM primal first phase.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Force traditional dual first phase.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Force BigM dual first phase.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Always use artificial bounds in dual.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Use original problem basis only when warmstarting the KKT.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Skip the primal bound flips for ranged primal (might cause more trouble than good if the bounds are very large).</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Also do the single pivot crash.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Do not apply aggressive perturbation in dual.</td>
</tr>
</tbody>
</table>

**quadraticUnshift (integer):** Determines whether an extra solution purification step is called after a solution found by the quadratic simplex (either primal or dual)

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>No purification step.</td>
</tr>
<tr>
<td>1</td>
<td>Always do the purification step.</td>
</tr>
</tbody>
</table>

**randomSeed (integer):** Sets the initial seed to use for the pseudo-random number generator in the Optimizer

The sequence of random numbers is always reset using the seed when starting a new optimization run.
Range: \([-\infty, \ldots, \infty]\)

Default: 1

**refineOps** (*integer*): Specifies when the solution refiner should be executed to reduce solution infeasibilities

The refiner will attempt to satisfy the target tolerances for all original linear constraints before presolve or scaling has been applied.

If neither the 7th nor 8th bit is set, the refiner will use the primal simplex if the primal violations are larger than the dual violations, otherwise it will use the dual simplex. If both the 7th and 8th bit are set then the refiner will split the problem into a primal feasible and dual feasible part, and solve the first with primal simplex and the second with dual simplex.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 19

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to refineOps_lpOptimal.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to refineOps_mipSolution.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to refineOps_mipNodeLp.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to refineOps_lpPresolve.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to refineOps_iterativeRefiner.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to refineOps_refinerPrecision.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to refineOps_refinerUsePrimal.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to refineOps_refinerUseDual.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to refineOps_mipFixGlobals.</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to refineOps_mipFixGlobalsTarget.</td>
</tr>
</tbody>
</table>

**refineOps_iterativeRefiner** (*boolean*): Apply the iterative refiner to refine the solution

See also refineOps.

Default: 0

**refineOps_lpOptimal** (*boolean*): Run the solution refiner on an optimal solution of a non-global problem

See also refineOps.

Default: 1

**refineOps_lpPresolve** (*boolean*): Run the solution refiner on an optimal solution before postsolve on a non-global problem

See also refineOps.

Default: 1

**refineOps_mipFixGlobals** (*boolean*): Refine MIP solutions such that rounding them keeps the problem feasible when reoptimized
See also `refineOps`.

Default: 0

**refineOps.mipFixGlobalsTarget** *(boolean)*: Attempt to refine MIP solutions such that rounding them keeps the problem feasible when reoptimized, but accept integers solutions even if refinement fails

See also `refineOps`.

Default: 0

**refineOps.mipNodeLp** *(boolean)*: Run the solution refiner on each node of the MIP search

See also `refineOps`.

Default: 0

**refineOps.mipSolution** *(boolean)*: Run the solution refiner when a new solution is found during a global search

The refiner will be applied to the presolved solution before any post-solve operations are applied.

See also `refineOps`.

Default: 1

**refineOps.refinerPrecision** *(boolean)*: Use higher precision in the iterative refinement

See also `refineOps`.

Default: 0

**refineOps.refinerUseDual** *(boolean)*: If set, the iterative refiner will use the dual simplex algorithm

See also `refineOps`.

Default: 0

**refineOps.refinerUsePrimal** *(boolean)*: If set, the iterative refiner will use the primal simplex algorithm

See also `refineOps`.

Default: 0

**reform** *(boolean)*: Substitute out objective var and equ when possible

Default: 1

**relaxTreeMemoryLimit** *(real)*: When the memory used by the branch and bound search tree exceeds the target specified by the `treeMemoryLimit` control, the optimizer will try to reduce this by writing nodes to the global file
In rare cases, usually where the solve has many millions of very small nodes, the tree structural
data (which cannot be written to the global file) will grow large enough to approach or exceed the tree's memory target. When this happens, optimizer performance can degrade greatly as the solver makes heavy use of the global file in preference to memory. To prevent this, the solver will automatically relax the tree memory limit when it detects this case; the relaxTreeMemoryLimit control specifies the proportion of the previous memory limit by which to relax it. Set relaxTreeMemoryLimit to 0.0 to force the Xpress Optimizer to never relax the tree memory limit in this way.

While setting higher values of relaxTreeMemoryLimit can improve performance significantly for a small number of models in low memory situations, the user is advised to use the treeMemoryLimit control to tune the memory usage of the branch and bound tree, according to the solve characteristics of their problem, rather than increasing relaxTreeMemoryLimit.

Range: \([0, \infty]\]

Default: 0.1

relPivotTol (real): Simplex: Minimum size of pivot element relative to largest element in column

At each iteration a pivot element is chosen within a given column of the matrix. The relative pivot tolerance, relPivotTol, is the size of the element chosen relative to the largest possible pivot element in the same column.

Range: \([0, \infty]\]

Default: 1e-06

repairIndefInitEq (boolean): Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Repair if possible.</td>
</tr>
<tr>
<td>1</td>
<td>Do not repair.</td>
</tr>
</tbody>
</table>

reRun (boolean): Rerun with primal simplex when not optimal/feasible

Applies only in cases where presolve is turned on and the model is diagnosed as infeasible or unbounded. If rerun is nonzero, we rerun the model using primal simplex with presolve turned off in hopes of getting better diagnostic information. If rerun is zero, no good diagnostic information exists, so we return no solution, only an indication of unboundedness/infeasibility.

Default: 0

reslim (real): Overrides GAMS reslim option

Sets the resource limit. When the solver has used more than this amount of CPU time (in seconds) the system will stop the search and report the best solution found so far. maxTime overwrites this option.

resourceStrategy (boolean): Controls whether the optimizer is allowed to make nondeterministic decisions if memory is running low in an effort to preserve memory and finish the solve

Available memory (or container limits) are automatically detected but can also be changed by maxMemorySoft and maxMemoryHard.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Allow the optimizer to change the solve path if necessary to preserve memory when getting close to one of the memory limits.</td>
</tr>
</tbody>
</table>

**rootPresolve** *(integer)*: Determines if presolving should be performed on the problem after the global search has finished with root cutting and heuristics

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Let the optimizer decide if the problem should be presolved again.</td>
</tr>
<tr>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Always presolve the root problem.</td>
</tr>
</tbody>
</table>

**sbBest** *(integer)*: Number of infeasible global entities to initialize pseudo costs for on each node

By default, strong branching will be performed only for infeasible global entities whose pseudo costs have not otherwise been initialized (see **historyCosts**). If sbBest is set to zero, the control **historyCosts** will also be treated as zero and no past branching or strong branching information will be used in the global entity selection.

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>0</td>
<td>Disable strong branching.</td>
</tr>
<tr>
<td>n&gt;0</td>
<td>Perform strong branching on up to n entities at each node.</td>
</tr>
</tbody>
</table>

**sbEffort** *(real)*: Adjusts the overall amount of effort when using strong branching to select an infeasible global entity to branch on

**sbEffort** is used as a multiplier on other strong branching related controls, and affects the values used for **sbBest**, **sbSelect** and **sbIterLimit** when those are set to automatic.

Range: 

Default: 1

**sbEstimate** *(integer)*: Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible global entity to branch on

These pseudo costs are used in combination with local strong branching and history costs to select the branch candidate.

Default: **auto**

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>1-6</td>
<td>Different variants of local pseudo costs.</td>
</tr>
</tbody>
</table>

**sbIterLimit** *(integer)*: Number of dual iterations to perform the strong branching for each entity
This control can be useful to increase or decrease the amount of effort (and thus time) spent performing strong branching at each node. Setting \texttt{sbIterLimit=}0 will disable dual strong branch iterations. Instead, the entity at the head of the candidate list will be selected for branching.

Range: \{-1, ..., \infty\}

Default: auto

\texttt{sbSelect (integer): Size of the candidate list of global entities for strong branching $\leftarrow$}

Before strong branching is applied on a node of the branch and bound tree, a list of candidates is selected among the infeasible global entities. These entities are then evaluated based on the local LP solution and prioritized. Strong branching will then be applied to the \texttt{sbBest} candidates. The evaluation is potentially expensive and for some problems it might improve performance if the size of the candidate list is reduced.

Default: $-2$

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-2$</td>
<td>Automatic (low effort).</td>
</tr>
<tr>
<td>$-1$</td>
<td>Automatic (high effort).</td>
</tr>
<tr>
<td>$n \geq 0$</td>
<td>Include $n$ entities in the candidate list (but always at least \texttt{sbBest} candidates).</td>
</tr>
</tbody>
</table>

\texttt{scaling (integer): Determines how the Optimizer will rescale a model internally before optimization $\leftarrow$}

If set to 0, no scaling will take place.

Setting \texttt{scaling} to 0 will preserve the current scaling of the problem. Note that the Optimizer might automatically select a different scaling strategy, when the control \texttt{autoScaling} is not disabled. However, if \texttt{scaling} is set to any value by the user, \texttt{autoScaling} will be ignored.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 163

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to \texttt{scaling_rowScaling}.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to \texttt{scaling_colScaling}.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to \texttt{scaling_rowScalingAgain}.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to \texttt{scaling_maximum}.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to \texttt{scaling_curtisReid}.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to \texttt{scaling_byMaxElemNotGeoMean}.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to \texttt{scaling_bigM}.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to \texttt{scaling_simplexObjScaling}.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to \texttt{scaling_ignoreQuadRowPart}.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to \texttt{scaling_beforePresolve}.</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to \texttt{scaling_noScalingRowsUp}.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to \texttt{scaling_noScalingColsDown}.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to \texttt{scaling_disableGlobalObjScaling}.</td>
</tr>
<tr>
<td>bit 13 = 8192</td>
<td>Equivalent to \texttt{scaling_rhsScaling}.</td>
</tr>
<tr>
<td>bit 14 = 16384</td>
<td>Equivalent to \texttt{scaling_noAggressiveQScaling}.</td>
</tr>
<tr>
<td>bit 15 = 32768</td>
<td>Equivalent to \texttt{scaling_slackScaling}.</td>
</tr>
</tbody>
</table>
**scaling beforePresolve** *(boolean)*: Scale before presolve

See also **scaling**.

Default: 0

**scaling bigM** *(boolean)*: Treat big-M rows as normal rows

See also **scaling**.

Default: 0

**scaling byMaxElemNotGeoMean** *(boolean)*: 0: scale by geometric mean

1: scale by maximum element.

See also **scaling**.

Default: 1

**scaling colScaling** *(boolean)*: Column scaling

See also **scaling**.

Default: 1

**scaling curtisReid** *(boolean)*: Curtis-Reid

See also **scaling**.

Default: 0

**scaling disableGlobalObjScaling** *(boolean)*: Do not apply automatic global objective scaling

See also **scaling**.

Default: 0

**scaling ignoreQuadRowPart** *(boolean)*: Exclude the quadratic part of constraint when calculating scaling factors

See also **scaling**.

Default: 0

**scaling maximum** *(boolean)*: Maximum

See also **scaling**.

Default: 0

**scaling noAggressiveQScaling** *(boolean)*: Disable aggressive quadratic scaling

See also **scaling**.

Default: 0

**scaling noScalingColsDown** *(boolean)*: Do not scale columns down
See also scaling.

Default: 0

**scaling_noScalingRowsUp** *(boolean)*: Do not scale rows up →

See also scaling.

Default: 0

**scaling_rhsScaling** *(boolean)*: RHS scaling →

See also scaling.

Default: 0

**scaling_rowScaling** *(boolean)*: Row scaling →

See also scaling.

Default: 1

**scaling_rowScalingAgain** *(boolean)*: Row scaling again →

See also scaling.

Default: 0

**scaling_simplexObjScaling** *(boolean)*: Scale objective function for the simplex method →

See also scaling.

Default: 1

**scaling_slackScaling** *(boolean)*: Enable explicit linear slack scaling →

See also scaling.

Default: 0

**sifting** *(integer)*: Determines whether to enable sifting algorithm with the dual simplex method →

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatically determined.</td>
</tr>
<tr>
<td>0</td>
<td>Disable sifting.</td>
</tr>
<tr>
<td>1</td>
<td>Enable sifting.</td>
</tr>
</tbody>
</table>

**siftPasses** *(integer)*: Determines how quickly we allow to grow the worker problems during the sifting algorithm →

Using larger values can increase the number of columns added to the worker problem which often results in increased solve times for the worker problems but the number of necessary sifting iterations may be reduced.
Range: \( \{0, ..., \infty\} \)
Default: 4

**siftPresolveOps (integer)**: Determines the presolve operations for solving the subproblems during the sifting algorithm

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>Use the <code>presolveOps</code> setting specified for the original problem.</td>
</tr>
<tr>
<td>(\geq 0)</td>
<td>Use the value for the <code>presolveOps</code> parameter for solving the subproblems during the sifting algorithm.</td>
</tr>
</tbody>
</table>

**siftSwitch (integer)**: Determines which algorithm to use for solving the subproblems during sifting

Default: \(-1\)

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>Dual simplex.</td>
</tr>
<tr>
<td>0</td>
<td>Barrier.</td>
</tr>
<tr>
<td>(&gt;0)</td>
<td>Use the barrier algorithm while the number of dual infeasibilities is larger than this value, otherwise use dual simplex.</td>
</tr>
</tbody>
</table>

**sleepOnThreadWait (integer)**: Determines if the threads should be put into a wait state when waiting for work

Default: `auto`

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>Automatically determined depending on the CPU the Optimizer is running on.</td>
</tr>
<tr>
<td>0</td>
<td>Keep the threads busy when waiting for work.</td>
</tr>
<tr>
<td>1</td>
<td>Put the threads into a wait state when waiting for work.</td>
</tr>
</tbody>
</table>

**solnpool (string)**: Solution pool file name

If set, the integer feasible solutions generated during the global search will be saved to a solution pool. A GDX file whose name is given by this option will be created and will contain an index to separate GDX files containing the individual solutions in the solution pool.

Default: `none`

**solnpoolCapacity (integer)**: Limit on number of solutions to store

Range: \( \{1, ..., \infty\} \)
Default: 999999999

**solnpoolCullDiversity (integer)**: Cull \(N\) solutions based on solution diversity

When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the diversity of the solutions in the pool.

Range: \( \{-1, ..., \infty\} \)
Default: \(-1\)
solnpoolCullObj (integer): Cull N solutions based on objective values

When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the MIP objective function.

Range: \{-1, ..., \infty\}

Default: -1

solnpoolCullRounds (integer): Terminate solution generation after N culling rounds

Limits the rounds of culls performed due to a full solution pool.

Default: 999999999

solnpoolDupPolicy (integer): Policy to use when handling storage of duplicate solutions

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Keep all: All solutions are kept including duplicates.</td>
</tr>
<tr>
<td>1</td>
<td>Continuous: All variables are compared with an exact match. Duplicate solutions are discarded.</td>
</tr>
<tr>
<td>2</td>
<td>Discrete and continuous separate: Both the discrete component of a solution pair and the continuous solution variables are compared. The continuous variables are compared with an exact match. Duplicate solutions are discarded.</td>
</tr>
<tr>
<td>3</td>
<td>Discrete only: Only the discrete component of a solution pair is compared. Duplicate solutions are discarded.</td>
</tr>
</tbody>
</table>

solnpoolmerge (string): Solution pool file name for merged solutions

Default: none

solnpoolnumsym (integer): Maximum number of variable symbols when writing merged solutions

Range: \{1, ..., \infty\}

Default: 10

solnpoolPop (integer): Controls method used to populate the solution pool

By default the MIP solution pool merely stores the incumbent solutions that are found during the global search, without changing the behavior of the search itself. In contrast, the MIP solution enumerator makes it possible to enumerate all or many of the feasible solutions for the MIP, instead of searching for the best solution.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>generate solutions using the normal search algorithm</td>
</tr>
<tr>
<td>2</td>
<td>invoke the solution enumerator to generate solutions</td>
</tr>
</tbody>
</table>

solnpoolPrefix (string): File name prefix for GDX solution files
Default: `soln`

**solnpoolVerbosity** (integer): Controls verbosity of solution pool routines

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>no output</td>
</tr>
<tr>
<td>0</td>
<td>output only messages coming from the XPRESS libraries</td>
</tr>
<tr>
<td>1</td>
<td>add some messages logging the effect of solution pool options</td>
</tr>
<tr>
<td>2</td>
<td>debugging mode</td>
</tr>
</tbody>
</table>

**sosRefTol** (real): Minimum relative gap between the ordering values of elements in a special ordered set

The gap divided by the absolute value of the larger of the two adjacent values must be at least `sosRefTol`.

This tolerance must not be set lower than 1.0E-06.

Range: \([0, \infty)\]

Default: `1e-06`

**symmetry** (integer): Adjusts the overall amount of effort for symmetry detection

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No symmetry detection.</td>
</tr>
<tr>
<td>1</td>
<td>Conservative effort.</td>
</tr>
<tr>
<td>2</td>
<td>Intensive symmetry search.</td>
</tr>
</tbody>
</table>

**symSelect** (integer): Adjusts the overall amount of effort for symmetry detection

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Search the whole matrix (otherwise the 0, 1 and -1 coefficients only).</td>
</tr>
<tr>
<td>1</td>
<td>Search all entities (otherwise binaries only).</td>
</tr>
</tbody>
</table>

**threads** (integer): Default number of threads used during optimization

Controls the number of threads to use. Positive values will be compared to the number of available cores detected and reduced if greater than this amount. Non- positive values are interpreted as the number of cores to leave free so setting `threads` to 0 uses all available cores while setting `threads` to -1 leaves one core free for other tasks.

Range: \([-\infty, ..., \infty)\]

Default: 1
**trace (integer)**: Display the infeasibility diagnosis during presolve

If non-zero, an explanation of the logical deductions made by presolve to deduce infeasibility or unboundedness will be displayed on screen or sent to the message callback function.

Presolve is sometimes able to detect infeasibility and unboundedness in problems. The set of deductions made by presolve can allow the user to diagnose the cause of infeasibility or unboundedness in their problem. However, not all infeasibility or unboundedness can be detected and diagnosed in this way.

Range: \( \{0, \ldots, \infty\} \)

Default: 0

**treeCompression (integer)**: When writing nodes to the global file, the optimizer can try to use data-compression techniques to reduce the size of the global file on disk

The `treeCompression` control determines the strength of the data-compression algorithm used; higher values give superior data-compression at the affect of decreasing performance, while lower values compress quicker but not as effectively. Where `treeCompression` is set to 0, no data compression will be used on the global file.

Range: \( \{0, \ldots, \infty\} \)

Default: 2

**treeCoverCuts (integer)**: Branch and Bound: Number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree

Compare with the description for `coverCuts`. A value of -1 indicates the number of rounds is determined automatically.

Range: \( \{-1, \ldots, \infty\} \)

Default: `auto`

**treeCutSelect (integer)**: Bit vector providing detailed control of the cuts created during the tree search of a global solve

Use `cutSelect` to control cuts on the root node.

The default value is -1 which enables all bits. Any bits not listed in the above table should be left in their default 'on' state, since the interpretation of such bits might change in future versions of the optimizer.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <code>treeCutSelect_clique</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>treeCutSelect_mir</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>treeCutSelect_cover</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>treeCutSelect_mirRowAggregation</code>.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to <code>treeCutSelect_flowpath</code>.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to <code>treeCutSelect_implication</code>.</td>
</tr>
<tr>
<td>bit 13 = 8192</td>
<td>Equivalent to <code>treeCutSelect_liftAndProject</code>.</td>
</tr>
<tr>
<td>bit 14 = 16384</td>
<td>Equivalent to <code>treeCutSelect_disableCutRows</code>.</td>
</tr>
<tr>
<td>bit 15 = 32768</td>
<td>Equivalent to <code>treeCutSelect_gubCover</code>.</td>
</tr>
<tr>
<td>bit 16 = 65536</td>
<td>Equivalent to <code>treeCutSelect_zeroHalf</code>.</td>
</tr>
<tr>
<td>bit 17 = 131072</td>
<td>Equivalent to <code>treeCutSelect_indicator</code>.</td>
</tr>
</tbody>
</table>
treeCutSelect_clique (boolean): Clique cuts

See also treeCutSelect.

Default: 1

treeCutSelect_cover (boolean): Lifted cover cuts

See also treeCutSelect.

Default: 1

treeCutSelect_disableCutRows (boolean): Disable cutting from cut rows

See also treeCutSelect.

Default: 1

treeCutSelect_flowpath (boolean): Flow path cuts

See also treeCutSelect.

Default: 1

treeCutSelect_gubCover (boolean): Lifted GUB cover cuts

See also treeCutSelect.

Default: 1

treeCutSelect_implication (boolean): Implication cuts

See also treeCutSelect.

Default: 1

treeCutSelect_indicator (boolean): Indicator constraint cuts

See also treeCutSelect.

Default: 1

treeCutSelect_liftAndProject (boolean): Turn on automatic Lift and Project cutting strategy

See also treeCutSelect.

Default: 1

treeCutSelect_mir (boolean): Mixed Integer Rounding (MIR) cuts

See also treeCutSelect.

Default: 1

treeCutSelect_mirRowAggregation (boolean): Turn on row aggregation for MIR cuts

See also treeCutSelect.

Default: 1
treeCutSelect_zeroHalf (boolean): Zero-half cuts ←

See also treeCutSelect.

Default: 1

treeGomCuts (integer): Branch and Bound: Number of rounds of Gomory cuts generated at nodes other than the first node in the tree ←

Compare with the description for gomCuts. A value of -1 indicates the number of rounds is determined automatically.

Range: \{-1, ..., \infty\}

Default: auto

treeMemoryLimit (integer): Soft limit, in megabytes, for the amount of memory to use in storing the branch and bound search tree ←

This doesn’t include memory used for presolve, heuristics, solving the LP relaxation, etc. When set to 0 (the default), the optimizer will calculate a limit automatically based on the amount of free physical memory detected in the machine. When the memory used by the branch and bound tree exceeds this limit, the optimizer will try to reduce the memory usage by writing lower-rated sections of the tree to a file called the “global file”. Though the solve can continue if it cannot bring the tree memory usage below the specified limit, performance will be inhibited and a message will be printed to the log.

Range: \{0, ..., \infty\}

Default: auto

treeMemorySavingTarget (real): When the memory used by the branch-and-bound search tree exceeds the limit specified by the treeMemoryLimit control, the optimizer will try to save memory by writing lower-rated sections of the tree to the global file ←

The target amount of memory to save will be enough to bring memory usage back below the limit, plus enough extra to give the tree room to grow. The treeMemorySavingTarget control specifies the extra proportion of the tree’s size to try to save; for example, if the tree memory limit is 1000Mb and treeMemorySavingTarget is 0.1, when the tree size exceeds 1000Mb the optimizer will try to reduce the tree size to 900Mb. Reducing the value of treeMemorySavingTarget will cause less extra nodes of the tree to be written to the global file, but will result in the memory saving routine being triggered more often (as the tree will have less room in which to grow), which can reduce performance. Increasing the value of treeMemorySavingTarget will cause additional, more highly-rated nodes, of the tree to be written to the global file, which can cause performance issues if these nodes are required later in the solve.

Range: \[0, \infty\]

Default: 0.4

treeQCCuts (integer): Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for nodes other than the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation ←

This control only has an effect for problems with quadratic or second order cone constraints, and only if outer approximation has not been disabled by setting miqcpAlg to 0.

Range: \{-1, ..., \infty\}

Default: auto

varSelection (integer): Branch and Bound: Determines the formula used to calculate the estimate of each integer variable, and thus which integer variable is selected to be branched on at a given node ←

The variable selected to be branched on is the one with the maximum estimate.

Default: auto
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Determined automatically.</td>
</tr>
<tr>
<td>1</td>
<td>The minimum of the 'up' and 'down' pseudo costs.</td>
</tr>
<tr>
<td>2</td>
<td>The 'up' pseudo cost plus the 'down' pseudo cost.</td>
</tr>
<tr>
<td>3</td>
<td>The maximum of the 'up' and 'down' pseudo costs, plus twice the minimum of the 'up' and 'down' pseudo costs.</td>
</tr>
<tr>
<td>4</td>
<td>The maximum of the 'up' and 'down' pseudo costs.</td>
</tr>
<tr>
<td>5</td>
<td>The 'down' pseudo cost.</td>
</tr>
<tr>
<td>6</td>
<td>The 'up' pseudo cost.</td>
</tr>
<tr>
<td>7</td>
<td>A weighted combination of the 'up' and 'down' pseudo costs, where the weights depend on how fractional the variable is.</td>
</tr>
<tr>
<td>8</td>
<td>The product of the 'up' and 'down' pseudo costs.</td>
</tr>
</tbody>
</table>

**writePrtSol** *(boolean)*: Directs optimizer to output a ”printsol” file ←

Default: 0

**xslp_algorithm** *(integer)*: Bit map describing the SLP algorithm(s) to be used ←

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 166

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to xslp_algorithm_noStepBounds.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to xslp_algorithm_stepBoundsAsRequired.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to xslp_algorithm_estimateStepBounds.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to xslp_algorithm_dynamicDamping.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to xslp_algorithm_holdValues.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to xslp_algorithm_retainPreviousValue.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to xslp_algorithm_resetDeltaZ.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to xslp_algorithm_quickConvergenceCheck.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to xslp_algorithm_escalatePenalties.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to xslp_algorithm_switchToPrimal.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to xslp_algorithm_maxCostOption.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to xslp_algorithm_residualErrors.</td>
</tr>
<tr>
<td>bit 13 = 8192</td>
<td>Equivalent to xslp_algorithm_noLPPolishing.</td>
</tr>
<tr>
<td>bit 14 = 16384</td>
<td>Equivalent to xslp_algorithm_cascadeBounds.</td>
</tr>
<tr>
<td>bit 15 = 32768</td>
<td>Equivalent to xslp_algorithm_clampExtendedActiveSB.</td>
</tr>
<tr>
<td>bit 16 = 65536</td>
<td>Equivalent to xslp_algorithm_clampExtendedAll.</td>
</tr>
</tbody>
</table>

**xslp_algorithm_cascadeBounds** *(boolean)*: Step bounds are updated to accommodate cascaded values (otherwise cascaded values are pushed to respect step bounds) ←

Normally, cascading will respect the step bounds of the SLP variable being cascaded. However, allowing the cascaded value to fall outside the step bounds (i.e. expanding the step bounds) can lead to better linearizations, as cascading will set better values for the SLP variables regarding their determining rows; note, that this later strategy might interfere with convergence of the cascaded variables.

See also xslp_algorithm.
Default: 0

`xslp_algorithm_clampExtendedActiveSB (boolean):` Apply clamping when converged on extended criteria only with some variables having active step bounds

When clamping is applied, then in any iteration when the solution would normally be deemed converged on extended criteria only, an extra step bound shrinking step is applied to help imposing strict convergence. In this variant, clamping is only applied on variables that have converged on extended criteria only and have active step bounds.

See also `xslp_algorithm`.

Default: 0

`xslp_algorithm_clampExtendedAll (boolean):` Apply clamping when converged on extended criteria only

When clamping is applied, then in any iteration when the solution would normally be deemed converged on extended criteria only, an extra step bound shrinking step is applied to help imposing strict convergence. In this variant, clamping is applied on all variables that have converged on extended criteria only.

See also `xslp_algorithm`.

Default: 0

`xslp_algorithm_dynamicDamping (boolean):` Use dynamic damping

Dynamic damping is sometimes an alternative to step bounding as a means of encouraging convergence, but it does not have the same power to force convergence as do step bounds.

See also `xslp_algorithm`.

Default: 0

`xslp_algorithm_escalatePenalties (boolean):` Escalate penalties

Constraint penalties are increased after each SLP iteration where penalty vectors are present in the solution. Escalation applies an additional scaling factor to the penalty costs for active errors. This helps to prevent successive solutions becoming "stuck" because of a particular constraint, because its cost will be raised so that other constraints may become more attractive to violate instead and thus open up a new region to explore.

See also `xslp_algorithm`.

Default: 0

`xslp_algorithm_estimateStepBounds (boolean):` Estimate step bounds from early SLP iterations

If initial step bounds are not being explicitly provided, this gives a good method of calculating reasonable values. Values will tend to be larger rather than smaller, to reduce the risk of infeasibility caused by excessive tightness of the step bounds.

See also `xslp_algorithm`.

Default: 1

`xslp_algorithm_holdValues (boolean):` Do not update values which are converged within strict tolerance
Models which are numerically unstable may benefit from this setting, which does not update values which have effectively hardly changed. If a variable subsequently does move outside its strict convergence tolerance, it will be updated as usual.

See also `xslp_algorithm`.

Default: 0

**xslp_algorithm_maxCostOption** *(boolean): Continue optimizing after penalty cost reaches maximum* →

Normally if the penalty cost reaches its maximum (by default the value of infinity), the optimization will terminate with an unconverged solution. If the maximum value is set to a smaller value, then it may make sense to continue, using other means to determine when to stop.

See also `xslp_algorithm`.

Default: 0

**xslp_algorithm_noLPPolishing** *(boolean): Skip the solution polishing step if the LP postsolve returns a slightly infeasible, but claimed optimal solution* ↔

Due to the nature of the SLP linearizations, and in particular because of the large differences in the objective function (model objective against penalty costs) some dual reductions in the linear presolver might introduce numerically instable reductions that cause slight infeasibilities to appear in postsolve. It is typically more efficient to remove these infeasibilities with an extra call to the linear optimizer; compared to switching these reductions off, which usually has a significant cost in performance. This bit is provided for numerically very hard problems, when the polishing step proves to be too expensive (Xpress SLP will report these if any in the final log summary).

See also `xslp_algorithm`.

Default: 0

**xslp_algorithm_noStepBounds** *(boolean): Do not apply step bounds* ↔

The default algorithm uses step bounds to force convergence. Step bounds may not be appropriate if dynamic damping is used.

See also `xslp_algorithm`.

Default: 0

**xslp_algorithm_quickConvergenceCheck** *(boolean): Quick convergence check* ↔

Normally, each variable is checked against all convergence criteria until either a criterion is found which it passes, or it is declared "not converged". Later (extended convergence) criteria are more expensive to test and, once an unconverged variable has been found, the overall convergence status of the solution has been established. The quick convergence check carries out checks on the strict criteria, but omits checks on the extended criteria when an unconverged variable has been found.

See also `xslp_algorithm`.

Default: 1

**xslp_algorithm_resetDeltaZ** *(boolean): Reset xslp_delta_z to zero when converged and continue SLP* ↔
One of the mechanisms to avoid local optima is to retain small non-zero coefficients between delta vectors and constraints, even when the coefficient should strictly be zero. If this option is set, then a converged solution will be continued with zero coefficients as appropriate.

See also xslp_algorithm.

Default: 0

**xslp_algorithm_residualErrors (boolean)**: Accept a solution which has converged even if there are still significant active penalty error vectors

Normally, the optimization will continue if there are active penalty vectors in the solution. However, it may be that there is no feasible solution (and so active penalties will always be present). Setting bit 12 means that, if other convergence criteria are met, then the solution will be accepted as converged and the optimization will stop.

See also xslp_algorithm.

Default: 0

**xslp_algorithm_retainPreviousValue (boolean)**: Retain previous value when cascading if determining row is zero

If the determining row is zero (that is, all the coefficients interacting with it are either zero or in columns with a zero activity), then it is impossible to calculate a new value for the vector being cascaded. The choice is to use the solution value as it is, or to revert to the assumed value

See also xslp_algorithm.

Default: 1

**xslp_algorithm_stepBoundsAsRequired (boolean)**: Apply step bounds to SLP delta vectors only when required

Step bounds can be applied to all vectors simultaneously, or applied only when oscillation of the delta vector (change in sign between successive SLP iterations) is detected.

See also xslp_algorithm.

Default: 1

**xslp_algorithm_switchToPrimal (boolean)**: Use the primal simplex algorithm when all error vectors become inactive

The primal simplex algorithm often performs better than dual during the final stages of SLP optimization when there are relatively few basis changes between successive solutions. As it is impossible to establish in advance when the final stages are being reached, the disappearance of error vectors from the solution is used as a proxy.

See also xslp_algorithm.

Default: 0

**xslp_analyze (integer)**: Bit map activating additional options supporting model / solution path analysis

In most cases, the value of this control does not affect the solution process itself. However, bit 3 (extended summary) will cause SLP to do more function evaluations, and the presence of non-deterministic user functions might cause changes in the solution process. These options are off by default due to performance considerations.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to <code>xslp_analyze_extendedFinalSummary</code>.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to <code>xslp_analyze_infeasibleIteration</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>xslp_analyze_saveLinearizations</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>xslp_analyze_saveIterBasis</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>xslp_analyze_saveFile</code>.</td>
</tr>
</tbody>
</table>

`xslp_analyze_extendedFinalSummary` *(boolean)*: Include an extended iteration summary ↔
See also `xslp_analyze`.
Default: 0

`xslp_analyze_infeasibleIteration` *(boolean)*: Run infeasibility analysis on infeasible iterations ↔
See also `xslp_analyze`.
Default: 0

`xslp_analyze_saveFile` *(boolean)*: Create an Xpress SLP save file at every `xslp_autosave` iterations ↔
See also `xslp_analyze`.
Default: 0

`xslp_analyze_saveIterBasis` *(boolean)*: Write the initial basis of the linearizations to disk at every `xslp_autosave` iterations ↔
See also `xslp_analyze`.
Default: 0

`xslp_analyze_saveLinearizations` *(boolean)*: Write the linearizations to disk at every `xslp_autosave` iterations ↔
See also `xslp_analyze`.
Default: 0

`xslp_aTol_a` *(real)*: Absolute delta convergence tolerance ↔
The absolute delta convergence criterion assesses the change in value of a variable (δX) against the absolute delta convergence tolerance. If δX < `xslp_aTol_a` then the variable has converged on the absolute delta convergence criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.
Range: \([-\infty, \infty]\]
Default: `auto`

`xslp_aTol_r` *(real)*: Relative delta convergence tolerance ↔
The relative delta convergence criterion assesses the change in value of a variable (δX) relative to the value of the variable (X), against the relative delta convergence tolerance. If δX < X * `xslp_aTol_r` then the variable has converged on the relative delta convergence criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.
Range: \([-\infty, \infty]\]
Default: `auto`

`xslp_augmentation` *(integer)*: Bit map describing the SLP augmentation method(s) to be used ↔
Setting single boolean options will overwrite the single bits of this bit map option.
Default: 12
### Table: Value and Meaning of Bit Positions

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_minimum</code>.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_evenHanded</code>.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_equalityErrorVectors</code>.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_allErrorVectors</code>.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_penaltyDeltaVectors</code>.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_aMeanWeight</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_sbFromValues</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_sbFromAbsValues</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_stepBoundRows</code>.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_allRowErrorVectors</code>.</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td></td>
<td>Equivalent to <code>xslp_augmentation_noUpdateIfOnlyIV</code>.</td>
</tr>
</tbody>
</table>

**xslp_augmentation_allErrorVectors (boolean):** Penalty error vectors on all non-linear inequality constraints

The linearization of a nonlinear constraint is inevitably an approximation and so may not be feasible except at the point of linearization. Adding penalty error vectors allows the linear approximation to be violated at a cost and so ensures that the linearized constraint is feasible.

See also `xslp_augmentation`.

Default: 1

**xslp_augmentation_allRowErrorVectors (boolean):** Penalty error vectors on all constraints

If the linear portion of the underlying model may actually be infeasible, then applying penalty vectors to all rows may allow identification of the infeasibility and may also allow a useful solution to be found.

See also `xslp_augmentation`.

Default: 0

**xslp_augmentation_aMeanWeight (boolean):** Use arithmetic means to estimate penalty weights

Penalty weights are estimated from the magnitude of the elements in the constraint or interacting rows. Geometric means are normally used, so that a few excessively large or small values do not distort the weights significantly. Arithmetic means will value the coefficients more equally.

See also `xslp_augmentation`.

Default: 0

**xslp_augmentation_equalityErrorVectors (boolean):** Penalty error vectors on all non-linear equality constraints

The linearization of a nonlinear equality constraint is inevitably an approximation and so will not generally be feasible except at the point of linearization. Adding penalty error vectors allows the linear approximation to be violated at a cost and so ensures that the linearized constraint is feasible.

See also `xslp_augmentation`.

Default: 1
xslp_augmentation_evenHanded (boolean): Even handed augmentation ↔

Standard augmentation treats variables which appear in non-constant coefficients in a different way from those which contain non-constant coefficients. Even-handed augmentation treats them all in the same way by replacing each non-constant coefficient $C$ in a vector $V$ by a new coefficient $C \times V$ in the “equals” column (which has a fixed activity of 1) and creating delta vectors for all types of variable in the same way.

See also xslp_augmentation.

Default: 0

xslp_augmentation_minimum (boolean): Minimum augmentation ↔

Standard augmentation includes delta vectors for all variables involved in nonlinear terms (in non-constant coefficients or as vectors containing non-constant coefficients). Includes delta vectors only for variables in non-constant coefficients. This produces a smaller linearization, but there is less control on convergence, because convergence control (for example, step bounding) cannot be applied to variables without deltas.

See also xslp_augmentation.

Default: 0

xslp_augmentation_noUpdateIfOnlyIV (boolean): Initial values do not imply an SLP variable ↔

Having an initial value will not cause the augmentation to include the corresponding delta variable; i.e. treat the variable as an SLP variable. Useful to provide initial values necessary in the first linearization in case of a minimal augmentation, or as a convenience option when it’s easiest to set an initial value for all variables for some reason.

See also xslp_augmentation.

Default: 0

xslp_augmentation_penaltyDeltaVectors (boolean): Penalty vectors to exceed step bounds ↔

Although it has rarely been found necessary or desirable in practice, Xpress-SLP allows step bounds to be violated at a cost. This may help with feasibility but it generally slows down or prevents convergence, so it should be used only if found absolutely necessary.

See also xslp_augmentation.

Default: 0

xslp_augmentation_sbFromAbsValues (boolean): Estimate step bounds from absolute values of row coefficients ↔

If step bounds are to be imposed from the start, the best approach is to provide explicit values for the bounds. Alternatively, Xpress-SLP can estimate the values from the largest estimated magnitude of the coefficients in the relevant rows.

See also xslp_augmentation.

Default: 0

xslp_augmentation_sbFromValues (boolean): Estimate step bounds from values of row coefficients ↔
If step bounds are to be imposed from the start, the best approach is to provide explicit values for the bounds. Alternatively, Xpress-SLP can estimate the values from the range of estimated coefficient sizes in the relevant rows.

See also `xslp_augmentation`.

Default: 0

**xslp_augmentation_stepBoundRows** *(boolean)*: Row-based step bounds ←

Step bounds are normally applied as bounds on the delta variables. Some applications may find that using explicit rows to bound the delta vectors gives better results.

See also `xslp_augmentation`.

Default: 0

**xslp_autosave** *(integer)*: Frequency with which to save the model ←

A value of zero means that the model will not automatically be saved. A positive value of n will save model information at every nth SLP iteration as requested by `xslp_analyze`.

Range: \{0, ..., \infty\}

Default: 0

**xslp_barCrossOverStart** *(integer)*: Default crossover activation behaviour for barrier start ←

When `xslp_barLimit` is set, `xslp_barCrossOverStart` offers an overwrite control on when crossover is applied. A positive value indicates that crossover should be disabled in iterations smaller than `xslp_barCrossOverStart` and should be enabled afterwards, or when stalling is detected as described in `xslp_barStartOps`. A value of 0 indicates to respect the value of crossover and only overwrite its value when stalling is detected. A value of -1 indicates to always rely on the value of crossover.

Range: \{0, ..., \infty\}

Default: 0

**xslp_barLimit** *(integer)*: Number of initial SLP iterations using the barrier method ←

Particularly for larger models, using the Newton barrier method is faster in the earlier SLP iterations. Later on, when the basis information becomes more useful, a simplex method generally performs better. `xslp_barLimit` sets the number of SLP iterations which will be performed using the Newton barrier method.

Range: \{0, ..., \infty\}

Default: 0

**xslp_barStallingLimit** *(integer)*: Number of iterations to allow numerical failures in barrier before switching to dual ←

On large problems, it may be beneficial to warm start progress by running a number of iterations with the barrier solver as specified by `xslp_barLimit`. On some numerically difficult problems, the barrier may stop prematurely due to numerical issues. Such solves can sometimes be finished if crossover is applied. After `xslp_barStallingLimit` such attempts, SLP will automatically switch to use the dual simplex.

Range: \{0, ..., \infty\}

Default: 3
**xslp_barStallingObjLimit** *(integer)*: Number of iterations over which to measure the objective change for barrier iterations with no crossover

On large problems, it may be beneficial to warm start progress by running a number of iterations with the barrier solver without crossover by setting `xslp_barLimit` to a positive value and setting `crossover` to 0. A potential drawback is slower convergence due to the interior point provided by the barrier solve keeping a higher number of variables active. This may lead to stalling in progress, negating the benefit of using the barrier. When in the last `xslp_barStallingObjLimit` iterations no significant progress has been made, crossover is automatically enabled.

Range: \{0, ..., \infty\}

Default: 3

**xslp_barStallingTol** *(real)*: Required change in the objective when progress is measured in barrier iterations without crossover

Minimum objective variability change required in relation to control `xslp_barStallingObjLimit` for the iterations to be regarded as making progress. The net objective, error cost and error sum are taken into account.

Range: [0, \infty]

Default: 0.05

**xslp_barStartOps** *(integer)*: Controls behaviour when the barrier is used to solve the linearizations

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to <code>xslp_barStartOps_stallingObjective</code>.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to <code>xslp_barStartOps_stallingNumerical</code>.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to <code>xslp_barStartOps_allowInteriorSol</code>.</td>
</tr>
</tbody>
</table>

**xslp_barStartOps_allowInteriorSol** *(boolean)*: If a non-vertex converged solution found by barrier without crossover can be returned as a final solution

See also `xslp_barStartOps`.

Default: 1

**xslp_barStartOps_stallingNumerical** *(boolean)*: Fall back to dual simplex if too many numerical problems are reported by the barrier

See also `xslp_barStartOps`.

Default: 1

**xslp_barStartOps_stallingObjective** *(boolean)*: Check objective progress when no crossover is applied

See also `xslp_barStartOps`.

Default: 1
\textbf{xslp\_calcThreads} \textit{(integer)}: Number of threads used for formula and derivatives evaluations

When beneficial, SLP can calculate formula values and partial derivative information in parallel.

Range: \{-1, ..., 1\}

Default: \texttt{auto}

\textbf{xslp\_cdTol\_a} \textit{(real)}: Absolute tolerance for deducing constant derivatives

The absolute tolerance test for constant derivatives is used as follows: If the value of the user function at point X0 is Y0 and the values at (X0-\(\delta X\)) and (X0+\(\delta X\)) are Yd and Yu respectively, then the numerical derivatives at X0 are: "down" derivative \(Dd = (Y0 - Yd) / \delta X\) "up" derivative \(Du = (Yu - Y0) / \delta X\) If abs(Dd-Du) \(\leq\) xslp\_cdTol\_a then the derivative is regarded as constant.

Range: \([0, \infty)\]

Default: \texttt{1e-08}

\textbf{xslp\_cdTol\_r} \textit{(real)}: Relative tolerance for deducing constant derivatives

The relative tolerance test for constant derivatives is used as follows: If the value of the user function at point X0 is Y0 and the values at (X0-\(\delta X\)) and (X0+\(\delta X\)) are Yd and Yu respectively, then the numerical derivatives at X0 are: "down" derivative \(Dd = (Y0 - Yd) / \delta X\) "up" derivative \(Du = (Yu - Y0) / \delta X\) If abs(Dd-Du) \(\leq\) xslp\_cdTol\_r * abs(Yd+Yu)/2 then the derivative is regarded as constant.

Range: \([0, \infty)\]

Default: \texttt{1e-08}

\textbf{xslp\_clampShrink} \textit{(real)}: Shrink ratio used to impose strict convergence on variables converged in extended criteria only

If the solution has converged but there are variables converged on extended criteria only, the xslp\_clampShrink acts as a shrinking ratio on the step bounds and the problem is optimized (if necessary multiple times), with the purpose of expediting strict convergence on all variables. xslp\_algorithm controls if this shrinking is applied at all, and if shrinking is applied to if the variables converged on extended criteria only with active step bounds only, or if on all variables.

Range: \([0, 1]\]

Default: \texttt{0.3}

\textbf{xslp\_clampValidationTol\_a} \textit{(real)}: Absolute validation tolerance for applying xslp\_clampShrink

If set and the absolute validation value is larger than this value, then control xslp\_clampShrink is checked once the solution has converged, but there are variables converged on extended criteria only.

Range: \([0, \infty]\)

Default: \texttt{not set}

\textbf{xslp\_clampValidationTol\_r} \textit{(real)}: Relative validation tolerance for applying xslp\_clampShrink

If set and the relative validation value is larger than this value, then control xslp\_clampShrink is checked once the solution has converged, but there are variables converged on extended criteria only.

Range: \([0, \infty]\)

Default: \texttt{not set}

\textbf{xslp\_convergenceOps} \textit{(integer)}: Bit map describing which convergence tests should be carried out

Provides fine tuned control (over setting the related convergence tolerances) of which convergence checks are carried out.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: \texttt{7167}
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to xslp_convergenceOps_cTol.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to xslp_convergenceOps_aTol.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to xslp_convergenceOps_mTol.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to xslp_convergenceOps_iTol.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to xslp_convergenceOps_sTol.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Check for user provided convergence.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to xslp_convergenceOps_vTol.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to xslp_convergenceOps_xTol.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to xslp_convergenceOps_oTol.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to xslp_convergenceOps_wTol.</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to xslp_convergenceOps_extendedScaling.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to xslp_convergenceOps_validation.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to xslp_convergenceOps_validationK.</td>
</tr>
</tbody>
</table>

**xslp_convergenceOps_aTol** *(boolean)*: Execute the delta tolerance checks

See also *xslp_convergenceOps*.

Default: 1

**xslp_convergenceOps_cTol** *(boolean)*: Execute the closure tolerance checks

See also *xslp_convergenceOps*.

Default: 1

**xslp_convergenceOps_extendedScaling** *(boolean)*: Take scaling of individual variables / rows into account

See also *xslp_convergenceOps*.

Default: 0

**xslp_convergenceOps_iTol** *(boolean)*: Execute the impact tolerance checks

See also *xslp_convergenceOps*.

Default: 1

**xslp_convergenceOps_mTol** *(boolean)*: Execute the matrix tolerance checks

See also *xslp_convergenceOps*.

Default: 1

**xslp_convergenceOps_oTol** *(boolean)*: Execute the objective range + active step bound check

See also *xslp_convergenceOps*.

Default: 1

**xslp_convergenceOps_sTol** *(boolean)*: Execute the slack impact tolerance checks


See also `xslp_convergenceOps`.

Default: 1

`xslp_convergenceOps_validation (boolean): Execute the validation target convergence checks ←`  
See also `xslp_convergenceOps`.

Default: 1

`xslp_convergenceOps_validationK (boolean): Execute the first order optimality target convergence checks ←`  
See also `xslp_convergenceOps`.

Default: 1

`xslp_convergenceOps_vTol (boolean): Execute the objective range checks ←`  
See also `xslp_convergenceOps`.

Default: 1

`xslp_convergenceOps_wTol (boolean): Execute the convergence continuation check ←`  
See also `xslp_convergenceOps`.

Default: 1

`xslp_convergenceOps_xTol (boolean): Execute the objective range + constraint activity check ←`  
See also `xslp_convergenceOps`.

Default: 1

`xslp_cTol (real): Closure convergence tolerance ←`

The closure convergence criterion measures the change in value of a variable ($\delta X$) relative to the value of its initial step bound (B), against the closure convergence tolerance. If $\delta X < B \times xslp_cTol$ then the variable has converged on the closure convergence criterion. If no explicit initial step bound is provided, then the test will not be applied and the variable can never converge on the closure criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: $[-\infty, \infty]$  

Default: auto

`xslp_cutStrategy (integer): Determines whihc cuts to apply in the MISLP search when the default SLP-in-MIP strategy is used ←`  

Cuts are derived from the linearizations and are local cuts in that they are valid in the linearization and not necessarily valid for the full problem. The values mirror that of `XPRS_CUTSTRATEGY`.

Range: $\{-1, ..., 3\}$  

Default: 0
**xslp_damp (real):** Damping factor for updating values of variables

The damping factor sets the next assumed value for a variable based on the previous assumed value (X0) and the actual value (X1). The new assumed value is given by \(X1 \times xslp_damp + X0 \times (1-xslp_damp)\)

Range: \([0, 1]\)

Default: 1

**xslp_dampExpand (real):** Multiplier to increase damping factor during dynamic damping

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are \(xslp_sameDamp\) successive changes in the same direction for a variable, then the damping factor \(D\) for the variable will be reset to \(D \times xslp_dampExpand + xslp_dampMax \times (1-xslp_dampExpand)\)

Range: \([0, 1]\)

Default: 1

**xslp_dampMax (real):** Maximum value for the damping factor of a variable during dynamic damping

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are \(xslp_sameDamp\) successive changes in the same direction for a variable, then the damping factor \(D\) for the variable will be reset to \(D \times xslp_dampExpand + xslp_dampMax \times (1-xslp_dampExpand)\)

Range: \([0, 1]\)

Default: 1

**xslp_dampMin (real):** Minimum value for the damping factor of a variable during dynamic damping

If dynamic damping is enabled, the damping factor for a variable will be decreased if successive changes are in the opposite direction. More precisely, the damping factor \(D\) for the variable will be reset to \(D \times xslp_dampShrink + xslp_dampMin \times (1-xslp_dampExpand)\)

Range: \([0, 1]\)

Default: 1

**xslp_dampShrink (real):** Multiplier to decrease damping factor during dynamic damping

If dynamic damping is enabled, the damping factor for a variable will be decreased if successive changes are in the opposite direction. More precisely, the damping factor \(D\) for the variable will be reset to \(D \times xslp_dampShrink + xslp_dampMin \times (1-xslp_dampExpand)\)

Range: \([0, 1]\)

Default: 1

**xslp_dampStart (integer):** SLP iteration at which damping is activated

If damping is used as part of the SLP algorithm, it can be delayed until a specified SLP iteration. This may be appropriate when damping is used to encourage convergence after an un-damped algorithm has failed to converge.

Range: \(\{0, ..., \infty\}\)

Default: 0
**xslp_defaultStepBound** (real): Minimum initial value for the step bound of an SLP variable if none is explicitly given

If no initial step bound value is given for an SLP variable, this will be used as a minimum value. If the algorithm is estimating step bounds, then the step bound actually used for a variable may be larger than the default. A default initial step bound is ignored when testing for the closure tolerance xslp.cTol: if there is no specific value, then the test will not be applied.

Range: \([0, \infty]\)

Default: 16

**xslp_deltaCost** (real): Initial penalty cost multiplier for penalty delta vectors

If penalty delta vectors are used, this parameter sets the initial cost factor. If there are active penalty delta vectors, then the penalty cost may be increased.

Range: \([0, \infty]\)

Default: 200

**xslp_deltaCostFactor** (real): Factor for increasing cost multiplier on total penalty delta vectors

If there are active penalty delta vectors, then the penalty cost multiplier will be increased by a factor of xslp_deltaCostFactor up to a maximum of xslp_deltaMaxCost

Range: \([1, \infty]\)

Default: 1.3

**xslp_deltaMaxCost** (real): Maximum penalty cost multiplier for penalty delta vectors

If there are active penalty delta vectors, then the penalty cost multiplier will be increased by a factor of xslp_deltaCostFactor up to a maximum of xslp_deltaMaxCost

Range: \([0, \infty]\)

Default: 1e+20

**xslp_deltaZLimit** (integer): Number of SLP iterations during which to apply xslp_delta.z

xslp_delta.z is used to retain small derivatives which would otherwise be regarded as zero. This is helpful in avoiding local optima, but may make the linearized problem more difficult to solve because of the number of small nonzero elements in the resulting matrix. xslp_deltaZLimit can be set to a nonzero value, which is then the number of iterations for which xslp_delta.z will be used. After that, small derivatives will be set to zero. A negative value indicates no automatic perturbations to the derivatives in any situation.

Range: \(\{0, \ldots, \infty\}\)

Default: 0

**xslp_delta_a** (real): Absolute perturbation of values for calculating numerical derivatives
First-order derivatives are calculated by perturbing the value of each variable in turn by a small amount. The amount is determined by the absolute and relative delta factors as follows: \( \text{xslp}_\text{delta}_a + \text{abs}(X) \times \text{xslp}_\text{delta}_r \) where \( X \) is the current value of the variable. If the perturbation takes the variable outside a bound, then the perturbation normally made only in the opposite direction.

Range: \([0, \infty]\)
Default: 0.001

**xslp\_delta\_r** *(real):* Relative perturbation of values for calculating numerical derivatives

First-order derivatives are calculated by perturbing the value of each variable in turn by a small amount. The amount is determined by the absolute and relative delta factors as follows: \( \text{xslp}_\text{delta}_a + \text{abs}(X) \times \text{xslp}_\text{delta}_r \) where \( X \) is the current value of the variable. If the perturbation takes the variable outside a bound, then the perturbation normally made only in the opposite direction.

Range: \([0, \infty]\)
Default: 0.001

**xslp\_delta\_x** *(real):* Minimum absolute value of delta coefficients to be retained

If the value of a coefficient in a delta column is less than this value, it will be reset to zero. Larger values of \( \text{xslp}_\text{delta}_x \) will result in matrices with fewer elements, which may be easier to solve. However, there will be increased likelihood of local optima as some of the small relationships between variables and constraints are deleted. There may also be increased difficulties with singular bases resulting from deletion of pivot elements from the matrix.

Range: \([0, \infty]\)
Default: \(1e-06\)

**xslp\_delta\_z** *(real):* Tolerance used when calculating derivatives

If the absolute value of a variable is less than this value, then a value of \( \text{xslp}_\text{delta}_z \) will be used instead for calculating derivatives. If a nonzero derivative is calculated for a formula which always results in a matrix coefficient less than \( \text{xslp}_\text{delta}_z \), then a larger value will be substituted so that at least one of the coefficients is \( \text{xslp}_\text{delta}_z \) in magnitude. If \( \text{xslp}_\text{deltaZLimit} \) is set to a positive number, then when that number of iterations have passed, values smaller than \( \text{xslp}_\text{delta}_z \) will be set to zero.

Range: \([0, \infty]\)
Default: \(1e-06\)

**xslp\_delta\_zero** *(real):* Absolute zero acceptance tolerance used when calculating derivatives

Provides an override value for the \( \text{xslp}_\text{delta}_z \) behavior. Derivatives smaller than \( \text{xslp}_\text{delta}_\text{zero} \) will not be substituted by \( \text{xslp}_\text{delta}_z \), defining a range in which derivatives are deemed nonzero and are affected by \( \text{xslp}_\text{delta}_z \). A negative value means that this tolerance will not be applied.

Range: \([-\infty, \infty]\)
Default: -1

**xslp\_derivatives** *(boolean):* Bitmap describing the method of calculating derivatives

If no bits are set then numerical derivatives are calculated using finite differences. Analytic derivatives cannot be used for formulae involving discontinuous functions. They may not work well with functions which are not smooth (such as \( \text{MAX} \)), or where the derivative changes very quickly with the value of the variable (such as \( \text{LOG} \) of small values). Both first and second order analytic derivatives can either be calculated as symbolic formulas, or by the means of auto-differentiation, with the exception that the second order symbolic derivatives require that the first order derivatives are also calculated using the symbolic method.

Default: 1
### xslp\_djTol (real):
Tolerance on DJ value for determining if a variable is at its step bound 🔄

If a variable is at its step bound and within the absolute delta tolerance \( xslp\_aTol\_a \) or closure tolerance \( xslp\_cTol\) then the step bounds will not be further reduced. If the DJ is greater in magnitude than \( xslp\_djTol \) then the step bound may be relaxed if it meets the necessary criteria.

Range: \([0, \infty)\]

Default: \(1e-06\)

### xslp\_ecfCheck (integer):
Check feasibility at the point of linearization for extended convergence criteria 🔄

The extended convergence criteria measure the accuracy of the solution of the linear approximation compared to the solution of the original nonlinear problem. For this to work, the linear approximation needs to be reasonably good at the point of linearization. In particular, it needs to be reasonably close to feasibility. \( xslp\_ecfCheck \) is used to determine what checking of feasibility is carried out at the point of linearization. If the point of linearization at the start of an SLP iteration is deemed to be infeasible, then the extended convergence criteria are not used to decide convergence at the end of that SLP iteration. If all that is required is to decide that the point of linearization is not feasible, then the search can stop after the first infeasible constraint is found (parameter is set to 1). If the actual number of infeasible constraints is required, then \( xslp\_ecfCheck \) should be set to 2, and all constraints will be checked. The number of infeasible constraints found at the point of linearization is returned in \( XSLP\_ECFCOUNT \).

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No check (extended criteria are always used);</td>
</tr>
<tr>
<td>1</td>
<td>Check until one infeasible constraint is found;</td>
</tr>
<tr>
<td>2</td>
<td>Check all constraints.</td>
</tr>
</tbody>
</table>

### xslp\_ecfTol\_a (real):
Absolute tolerance on testing feasibility at the point of linearization 🔄

The extended convergence criteria test how well the linearization approximates the true problem. They depend on the point of linearization being a reasonable approximation — in particular, that it should be reasonably close to feasibility. Each constraint is tested at the point of linearization, and the total positive and negative contributions to the constraint from the columns in the problem are calculated. A feasibility tolerance is calculated as the largest of \( xslp\_ecfTol\_a \) and \( \max(\text{abs(Positive)}, \text{abs(Negative)}) \) * \( xslp\_ecfTol\_r \) If the calculated infeasibility is greater than the tolerance, the point of linearization is regarded as infeasible and the extended convergence criteria will not be applied. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target \( xslp\_validationTarget\_r \). Good values for the control are usually fall between 1e-1 and 1e-6.

Range: \([-\infty, \infty]\)

Default: auto
**xslp_ecfTol_r (real):** Relative tolerance on testing feasibility at the point of linearization

The extended convergence criteria test how well the linearization approximates the true problem. They depend on the point of linearization being a reasonable approximation — in particular, that it should be reasonably close to feasibility. Each constraint is tested at the point of linearization, and the total positive and negative contributions to the constraint from the columns in the problem are calculated. A feasibility tolerance is calculated as the largest of xslp_ecfTol_a and \( \max(\text{abs}(\text{Positive}), \text{abs}(\text{Negative})) \times xslp_ecfTol_r \) If the calculated infeasibility is greater than the tolerance, the point of linearization is regarded as infeasible and the extended convergence criteria will not be applied. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-1 and 1e-6.

Range: \([-\infty, \infty]\]  
Default: auto

**xslp_enforceCostShrink (real):** Factor by which to decrease the current penalty multiplier when enforcing rows

When feasibility of a row cannot be achieved by increasing the penalty cost on its error variable, removing the variable (fixing it to zero) can force the row to be satisfied, as set by xslp_enforceMaxCost. After the error variables have been removed (which is equivalent to setting to row to be enforced) the penalties on the remaining error variables are rebalanced to allow for a reduction in the size of the penalties in the objective in order to achieve better numerical behaviour.

Range: \([0, 1]\]  
Default: 1e-05

**xslp_enforceMaxCost (real):** Maximum penalty cost in the objective before enforcing most violating rows

When feasibility of a row cannot be achieved by increasing the penalty cost on its error variable, removing the variable (fixing it to zero) can force the row to be satisfied. After the error variables have been removed (which is equivalent to setting to row to be enforced) the penalties on the remaining error variables are rebalanced to allow for a reduction in the size of the penalties in the objective in order to achieve better numerical behaviour, controlled by xslp_enforceCostShrink.

Range: \([0, \infty]\]  
Default: 1e+11

**xslp_errorCost (real):** Initial penalty cost multiplier for penalty error vectors

If penalty error vectors are used, this parameter sets the initial cost factor. If there are active penalty error vectors, then the penalty cost may be increased.

Range: \([0, \infty]\]  
Default: 200

**xslp_errorCostFactor (real):** Factor for increasing cost multiplier on total penalty error vectors
If there are active penalty error vectors, then the penalty cost multiplier will be increased by a factor of `xslp_errorCostFactor` up to a maximum of `xslp_errorMaxCost`.

Range: \([1, \infty]\)

Default: 1.3

**xslp_errorMaxCost (real):** Maximum penalty cost multiplier for penalty error vectors ←

If there are active penalty error vectors, then the penalty cost multiplier will be increased by a factor of `xslp_errorCostFactor` up to a maximum of `xslp_errorMaxCost`.

Range: \([0, \infty]\)

Default: 1e+20

**xslp_errorTol_a (real):** Absolute tolerance for error vectors ←

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than `xslp_errorTol_a`; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than `xslp_evTol_a`; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than `xslp_evTol_r * Obj` where Obj is the current objective function value.

Range: \([0, \infty]\)

Default: 1e-05

**xslp_errorTol_p (real):** Absolute tolerance for printing error vectors ←

The solution log includes a print of penalty delta and penalty error vectors with an activity greater than `xslp_errorTol_p`.

Range: \([0, \infty]\)

Default: 0.0001

**xslp_escalation (real):** Factor for increasing cost multiplier on individual penalty error vectors ←

If penalty cost escalation is activated in `xslp_algorithm` then the penalty cost multiplier will be increased by a factor of `xslp_escalation` for any active error vector up to a maximum of `xslp_maxWeight`.

Range: \([1, \infty]\)

Default: 1.25

**xslp_eTol_a (real):** Absolute tolerance on penalty vectors ←

For each penalty error vector, the contribution to its constraint is calculated, together with the total positive and negative contributions to the constraint from other vectors. If its contribution is less than `xslp_eTol_a` or less than `Positive * xslp_eTol_r` or less than `abs(Negative) * xslp_eTol_r` then it will be regarded as insignificant and will not have its penalty increased. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)

Default: 0.0001
xslp_eTol_r (real): Relative tolerance on penalty vectors

For each penalty error vector, the contribution to its constraint is calculated, together with the total positive and negative contributions to the constraint from other vectors. If its contribution is less than xslp_eTol_a or less than Positive*xslp_eTol_r or less than abs(Negative)*xslp_eTol_r then it will be regarded as insignificant and will not have its penalty increased. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]
Default: 0.0001

xslp_evTol_a (real): Absolute tolerance on total penalty costs

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than xslp_errorTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_r * Obj where Obj is the current objective function value. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-2 and 1e-6, but normally a magnitude larger than xslp_eTol_a.

Range: [-∞, ∞]
Default: -1

xslp_evTol_r (real): Relative tolerance on total penalty costs

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than xslp_errorTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_r * Obj where Obj is the current objective function value. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-2 and 1e-6, but normally a magnitude larger than xslp_eTol_r.

Range: [-∞, ∞]
Default: -1

xslp_expand (real): Multiplier to increase a step bound

If step bounding is enabled, the step bound for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameCount successive changes reaching the step bound and in the same direction for a variable, then the step bound (B) for the variable will be reset to B*xslp_expand.

Range: [1, ∞]
Default: 2

xslp_feasTolTarget (real): When set, this defines a target feasibility tolerance to which the linearizations are solved to
This is a soft version of XPRS_FEASTOL, and will dynamically revert back to XPRS_FEASTOL if the desired accuracy could not be achieved.

Range: \([0, \infty]\)

Default: not set

**xsplp_filter** *(integer)*: Bit map for controlling solution updates

Bits 0 determine if XSLPgetSlpsol should return the final converged solution, or the solution which had the best value according to the merit function. If bit 1 is set, a cascaded solution which does not improve the merit function will be rejected (Xpress SLP will revert to the solution of the linearization). Bits 2-3 determine the strategy for when the step direction is not improving according to the merit function.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 3

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to <strong>xsplp_filterKeepBest</strong>.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Check cascaded solutions against improvements in the merit function.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to <strong>xsplp_filterZeroLineSearch</strong>.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to <strong>xsplp_filterZeroLineSearchTR</strong>.</td>
</tr>
</tbody>
</table>

**xsplp_filterKeepBest** *(boolean)*: Retrain solution best according to the merit function

See also **xsplp_filter**.

Default: 1

**xsplp_filterZeroLineSearch** *(boolean)*: Force minimum step sizes in line search

See also **xsplp_filter**.

Default: 0

**xsplp_filterZeroLineSearchTR** *(boolean)*: Accept the trust region step is the line search returns a zero step size

See also **xsplp_filter**.

Default: 0

**xsplp_findIV** *(integer)*: Option for running a heuristic to find a feasible initial point

The procedure uses bound reduction (and, up to an extent, probing) to obtain a point in the initial bounding box that is feasible for the bound reduction techniques. If an initial point is already specified and is found not to violate bound reduction, then the heuristic is not run and the given point is used as the initial solution.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic (default).</td>
</tr>
<tr>
<td>0</td>
<td>Disable the heuristic.</td>
</tr>
<tr>
<td>1</td>
<td>Enable the heuristic.</td>
</tr>
</tbody>
</table>
**xslp_granularity** *(real)*: Base for calculating penalty costs

If `xslp_granularity > 1`, then initial penalty costs will be powers of `xslp_granularity`.

Range: `[1, ∞]`

Default: 4

**xslp_hessian** *(integer)*: Second order differentiation mode when using analytical derivatives

Symbolic mode differentiation for the second order derivatives is only available when `xslp_jacobian` is also set to symbolic mode.

Default: -1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numerical derivatives (finite difference)</td>
</tr>
<tr>
<td>2</td>
<td>Symbolic differentiation</td>
</tr>
<tr>
<td>3</td>
<td>Automatic differentiation</td>
</tr>
<tr>
<td>-1, 0</td>
<td>Automatic selection</td>
</tr>
</tbody>
</table>

**xslp_heurStrategy** *(integer)*: Branch and Bound: MINLP heuristic strategy

On some problems it is worth trying more comprehensive heuristic strategies by setting `HEURSTRATEGY` to 2 or 3. Note that `HEURSTRATEGY` is deprecated, use `heurEmphasis` instead.

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection of heuristic strategy.</td>
</tr>
<tr>
<td>0</td>
<td>No heuristics.</td>
</tr>
<tr>
<td>1</td>
<td>Basic heuristic strategy.</td>
</tr>
<tr>
<td>2</td>
<td>Enhanced heuristic strategy.</td>
</tr>
<tr>
<td>3</td>
<td>Extensive heuristic strategy.</td>
</tr>
<tr>
<td>4</td>
<td>Run all heuristics without effort limits.</td>
</tr>
</tbody>
</table>

**xslp_infeasLimit** *(integer)*: Maximum number of consecutive infeasible SLP iterations which can occur before Xpress-SLP terminates

An infeasible solution to an SLP iteration means that is likely that Xpress-SLP will create a poor linear approximation for the next SLP iteration. Sometimes, small infeasibilities arise because of numerical difficulties and do not seriously affect the solution process. However, if successive solutions remain infeasible, it is unlikely that Xpress-SLP will be able to find a feasible converged solution. `xslp_infeasLimit` sets the number of successive SLP iterations which must take place before Xpress-SLP terminates with a status of “infeasible solution”.

Range: `{0, ..., ∞}`

Default: 3

**xslp_infinity** *(real)*: Value returned by a divide-by-zero in a formula
Range: \([0, \infty]\)
Default: 1e+10

**xslp_iterLimit** (*integer*): Maximum number of SLP iterations

If Xpress-SLP reaches **xslp_iterLimit** without finding a converged solution, it will stop. For MISLP, the limit is on the number of SLP iterations at each node.

Range: \([-1, \ldots, \infty]\)
Default: auto

**xslp_sItol_a** (*real*): Absolute impact convergence tolerance

The absolute impact convergence criterion assesses the change in the effect of a coefficient in a constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. \(E = X \times C\) where \(X\) is the activity of the matrix column in which the coefficient appears, and \(C\) is the value of the coefficient. The linearization approximates the effect of the coefficient as \(E_1 = X \times C_0 + \delta X \times C'0\) where \(X\) is as before, \(C_0\) is the value of the coefficient \(C\) calculated using the assumed values for the variables and \(C'0\) is the value of \(\frac{\partial C}{\partial X}\) calculated using the assumed values for the variables. If \(C_1\) is the value of the coefficient \(C\) calculated using the actual values for the variables, then the error in the effect of the coefficient is given by \(\delta E = X \times C_1 - (X \times C_0 + \delta X \times C'0)\). If \(\delta E < xslp_sItol_a\) then the variable has passed the absolute impact convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target **xslp_validationTarget_r**. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)
Default: auto

**xslp_sItol_r** (*real*): Relative impact convergence tolerance

The relative impact convergence criterion assesses the change in the effect of a coefficient in a constraint in relation to the magnitude of the constituents of the constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. \(E = X \times C\) where \(X\) is the activity of the matrix column in which the coefficient appears, and \(C\) is the value of the coefficient. The linearization approximates the effect of the coefficient as \(E_1 = X \times C_0 + \delta X \times C'0\) where \(X\) is as before, \(C_0\) is the value of the coefficient \(C\) calculated using the assumed values for the variables and \(C'0\) is the value of \(\frac{\partial C}{\partial X}\) calculated using the assumed values for the variables. If \(C_1\) is the value of the coefficient \(C\) calculated using the actual values for the variables, then the error in the effect of the coefficient is given by \(\delta E = X \times C_1 - (X \times C_0 + \delta X \times C'0)\). All the elements of the constraint are examined, excluding delta and error vectors: for each, the contribution to the constraint is evaluated as the element multiplied by the activity of the vector in which it appears; it is then included in a total positive contribution or total negative contribution depending on the sign of the contribution. If the predicted effect of the coefficient is positive, it is tested against the total positive contribution; if the effect of the coefficient is negative, it is tested against the total negative contribution. If \(T_0\) is the total positive or total negative contribution to the constraint (as appropriate) and \(\delta E < T_0 xslp_sItol_r\) then the variable has passed the relative impact convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target **xslp_validationTarget_r**. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)
Default: auto
\textbf{xslp\_jacobian} \textit{(integer)}: First order differentiation mode when using analytical derivatives

Symbolic mode differentiation for the second order derivatives is only available when \textsl{xslp\_jacobian} is set to symbolic mode.

Default: \texttt{-1}

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numerical derivatives (finite difference)</td>
</tr>
<tr>
<td>2</td>
<td>Symbolic differentiation</td>
</tr>
<tr>
<td>3</td>
<td>Automatic differentiation</td>
</tr>
<tr>
<td>-1,0</td>
<td>Automatic selection</td>
</tr>
</tbody>
</table>

\textbf{xslp\_linQuadBR} \textit{(integer)}: Use linear and quadratic constraints and objective function to further reduce bounds on all variables

While bound reduction is effective when performed on nonlinear, nonquadratic constraints and objective function, it can be useful to obtain tightened bounds from linear and quadratic constraints, as the corresponding variables may appear in other nonlinear constraints. This option then allows for a slightly more expensive bound reduction procedure, at the benefit of further reduction in the problem’s bounds.

Default: \texttt{auto}

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection</td>
</tr>
<tr>
<td>0</td>
<td>Disable</td>
</tr>
<tr>
<td>1</td>
<td>Enable</td>
</tr>
</tbody>
</table>

\textbf{xslp\_log} \textit{(integer)}: Level of printing during SLP iterations

Default: \texttt{0}

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>Minimal</td>
</tr>
<tr>
<td>1</td>
<td>Normal: iteration, penalty vectors</td>
</tr>
<tr>
<td>2</td>
<td>Omit from convergence log any variables which have converged</td>
</tr>
<tr>
<td>3</td>
<td>Omit from convergence log any variables which have already converged (except variables on step bounds)</td>
</tr>
<tr>
<td>4</td>
<td>Include all variables in convergence log</td>
</tr>
<tr>
<td>5</td>
<td>Include user function call communications in the log</td>
</tr>
</tbody>
</table>

\textbf{xslp\_lsIterLimit} \textit{(integer)}: Number of iterations in the line search

The line search attempts to refine the step size suggested by the trust region step bounds. The line search is a local method; the control sets a maximum on the number of model evaluations during the line search.

Range: \{0, ..., \infty\}

Default: \texttt{0}
**xspl_{\text{lsPatternLimit}}** (integer): Number of iterations in the pattern search preceding the line search \( \leftrightarrow \)

When positive, defines the number of samples taken along the step size suggested by the trust region step bounds before initiating the line search. Useful for highly non-convex problems.

Range: \( \{0, ..., \infty\} \)

Default: 0

**xspl_{\text{lsStart}}** (integer): Iteration in which to active the line search \( \leftrightarrow \)

Range: \( \{0, ..., \infty\} \)

Default: 8

**xspl_{\text{lsZeroLimit}}** (integer): Maximum number of zero length line search steps before line search is deactivated \( \leftrightarrow \)

When the line search repeatedly returns a zero step size, counteracted by bits set on xspl_{filter}, the effort spent in line search is redundant, and line search will be deactivated after xspl_{lsZeroLimit} consecutive such iteration.

Range: \( \{0, ..., \infty\} \)

Default: 5

**xspl_{\text{maxWeight}}** (real): Maximum penalty weight for delta or error vectors \( \leftrightarrow \)

When penalty vectors are created, or when their weight is increased by escalation, the maximum weight that will be used is given by xspl_{maxWeight}.

Range: \( [0, \infty] \)

Default: 100

**xspl_{\text{meritLambda}}** (real): Factor by which the net objective is taken into account in the merit function \( \leftrightarrow \)

The merit function is evaluated in the original, non-augmented / linearized space of the problem. A solution is deemed improved, if either feasibility improved, or if feasibility is not deteriorated but the net objective is improved, or if the combination of the two is improved, where the value of the xspl_{meritLambda} control is used to combine the two measures. A nonpositive value indicates that the combined effect should not be checked.

Range: \( [0, \infty] \)

Default: 0

**xspl_{\text{minSBFactor}}** (real): Factor by which step bounds can be decreased beneath xspl_{aTol,a} \( \leftrightarrow \)

Normally, step bounds are not decreased beneath xspl_{aTol,a}, as such variables are treated as converged. However, it may be beneficial to decrease step bounds further, as individual variable value changes might affect the convergence of other variables in the model, even if the variable itself is deemed converged.

Range: \( [0, \infty] \)

Default: 1
**xslp\_minWeight** (*real*): Minimum penalty weight for delta or error vectors

When penalty vectors are created, the minimum weight that will be used is given by `xslp\_minWeight`.

Range: \([0, \infty]\)

Default: 0.01

**xslp\_mipAlgorithm** (*integer*): Bitmap describing the MISLP algorithms to be used

`xslp\_mipAlgorithm` determines the strategy of XSLP\_nlpoptimize for solving MINLP problems. The recommended approach is to solve the problem first without reference to the global variables. This can be handled automatically by setting bit 0 of `xslp\_mipAlgorithm`; if done manually, then optimize using the "l" option to prevent the Optimizer presolve from changing the problem. Some versions of the optimizer re-run the initial node as part of the global search; it is possible to initiate a new SLP optimization at this point by relaxing or fixing step bounds (use bits 2 and 3). If step bounds are fixed for a class of variable, then the variables in that class will not change their value in any child node. At each node, it is possible to relax or fix step bounds. It is recommended that step bounds are relaxed, so that the new problem can be solved starting from its parent, but without undue restrictions caused by step bounding (use bit 4). Exceptionally, it may be preferable to restrict the freedom of child nodes by relaxing fewer types of step bound or fixing the values of some classes of variable (use bit 5). When the optimal node has been found, it is possible to fix the global variables and then re-optimize with SLP. Step bounds can be relaxed or fixed for this optimization as well (use bits 7 and 8). Although it is ultimately necessary to solve the optimal node to convergence, individual nodes can be truncated after `xslp\_mipIterLimit` SLP iterations. Set bit 6 to activate this feature. The normal MISLP algorithm uses SLP at each node. One alternative strategy is to use the MIP optimizer for solving each SLP iteration. Set bit 9 to implement this strategy ("MIP within SLP"). Another strategy is to solve the problem to convergence ignoring the nature of the global variables. Then, fixing the linearization, use MIP to find the optimal setting of the global variables. Then, fixing the global variables, but varying the linearization, solve to convergence. Set bit 10 to implement this strategy ("SLP then MIP"). For more details about MISLP algorithms and strategies, see the separate section.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 17

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_initialSLP</code>.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_initialRelaxSLP</code>.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_initialFixSLP</code>.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_nodeRelaxSLP</code>.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_nodeFixSLP</code>.</td>
</tr>
<tr>
<td>bit 6 = 64</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_nodeLimitSLP</code>.</td>
</tr>
<tr>
<td>bit 7 = 128</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_finalRelaxSLP</code>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_finalFixSLP</code>.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_withinSLP</code>.</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to <code>xslp\_mipAlgorithm\_slpThenMIP</code>.</td>
</tr>
</tbody>
</table>

**xslp\_mipAlgorithm\_finalFixSLP** (*boolean*): Fix step bounds according to `xslp\_mipFixStepBounds` after MIP solution is found

See also `xslp\_mipAlgorithm`.
Default: 0

\texttt{xslp\_mipAlgorithm\_finalRelaxSLP} (boolean): Relax step bounds according to \texttt{xslp\_mipRelaxStepBounds} after MIP solution is found $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_initialFixSLP} (boolean): Fix step bounds according to \texttt{xslp\_mipFixStepBounds} after initial node $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_initialRelaxSLP} (boolean): Relax step bounds according to \texttt{xslp\_mipRelaxStepBounds} after initial node $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_initialSLP} (boolean): Solve initial SLP to convergence $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 1

\texttt{xslp\_mipAlgorithm\_nodeFixSLP} (boolean): Fix step bounds according to \texttt{xslp\_mipFixStepBounds} at each node $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_nodeLimitSLP} (boolean): Limit iterations at each node to \texttt{xslp\_mipIterLimit} $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_nodeRelaxSLP} (boolean): Relax step bounds according to \texttt{xslp\_mipRelaxStepBounds} at each node $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 1

\texttt{xslp\_mipAlgorithm\_slpThenMIP} (boolean): Use MIP on converged SLP solution and then SLP on the resulting MIP solution $\rightsquigarrow$

See also \texttt{xslp\_mipAlgorithm}.

Default: 0

\texttt{xslp\_mipAlgorithm\_withinSLP} (boolean): Use MIP at each SLP iteration instead of SLP at each node $\rightsquigarrow$
See also xslp.mipAlgorithm.

Default: 0

**xslp.mipCutOffCount (integer):** Number of SLP iterations to check when considering a node for cutting off

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp.mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp.mipCutOff_a and OBJ * xslp.mipCutOff_r where OBJ is the best integer solution obtained so far. The test is not applied until at least xslp.mipCutOffLimit SLP iterations have been carried out at the current node.

Range: {0, ..., ∞}

Default: 5

**xslp.mipCutOffLimit (integer):** Number of SLP iterations to check when considering a node for cutting off

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp.mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp.mipCutOff_a and OBJ * xslp.mipCutOff_r where OBJ is the best integer solution obtained so far. The test is not applied until at least xslp.mipCutOffLimit SLP iterations have been carried out at the current node.

Range: {0, ..., ∞}

Default: 10

**xslp.mipCutOff_a (real):** Absolute objective function cutoff for MIP termination

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp.mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp.mipCutOff_a and OBJ * xslp.mipCutOff_r where OBJ is the best integer solution obtained so far. The MIP cutoff tests are only applied after xslp.mipCutOffLimit SLP iterations at the current node.

Range: [0, ∞]

Default: 1e-05

**xslp.mipCutOff_r (real):** Absolute objective function cutoff for MIP termination

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp.mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp.mipCutOff_a and OBJ * xslp.mipCutOff_r where OBJ is the best integer solution obtained so far. The MIP cutoff tests are only applied after xslp.mipCutOffLimit SLP iterations at the current node.

Range: [0, ∞]

Default: 1e-05
xslp_mipDefaultAlgorithm (integer): Default algorithm to be used during the global search in MISLP

The default algorithm used within SLP during the MISLP optimization can be set using xslp_mipDefaultAlgorithm. It will not necessarily be the same as the one best suited to the initial SLP optimization.

Range: \( \{1, ..., 5\} \)

Default: auto

xslp_mipErrorTol_a (real): Absolute penalty error cost tolerance for MIP cut-off

The penalty error cost test is applied at each node where there are active penalties in the solution. If xslp_mipErrorTol_a is nonzero and the absolute value of the penalty costs is greater than xslp_mipErrorTol_a, the node will be declared infeasible. If xslp_mipErrorTol_a is zero then no test is made and the node will not be declared infeasible on this criterion.

Range: \([0, \infty]\)

Default: 0

xslp_mipErrorTol_r (real): Relative penalty error cost tolerance for MIP cut-off

The penalty error cost test is applied at each node where there are active penalties in the solution. If xslp_mipErrorTol_r is nonzero and the absolute value of the penalty costs is greater than xslp_mipErrorTol_r \( \times \) abs(Obj) where Obj is the value of the objective function, then the node will be declared infeasible. If xslp_mipErrorTol_r is zero then no test is made and the node will not be declared infeasible on this criterion.

Range: \([0, \infty]\)

Default: 0

xslp_mipFixStepBounds (integer): Bitmap describing the step-bound fixing strategy during MISLP

At any node (including the initial and optimal nodes) it is possible to fix the step bounds of classes of variables so that the variables themselves will not change. This may help with convergence, but it does increase the chance of a local optimum because of excessive artificial restrictions on the variables.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to xslp_mipFixStepBounds_structNotCoeff.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to xslp_mipFixStepBounds_structAll.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to xslp_mipFixStepBounds_coefOnly.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to xslp_mipFixStepBounds_coef.</td>
</tr>
</tbody>
</table>

xslp_mipFixStepBounds_coef (boolean): Fix step bounds on SLP variables appearing in coefficients

See also xslp_mipFixStepBounds.

Default: 0
xslp_mipFixStepBounds_coefOnly (boolean): Fix step bounds on SLP variables appearing only in coefficients

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipFixStepBounds_structAll (boolean): Fix step bounds on all structural SLP variables

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipFixStepBounds_structNotCoef (boolean): Fix step bounds on structural SLP variables which are not in coefficients

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipIterLimit (integer): Maximum number of SLP iterations at each node

If bit 6 of xslp_mipAlgorithm is set, then the number of iterations at each node will be limited to xslp_mipIterLimit.

Range: \{0, ..., \infty\}

Default: 0

xslp_mipLog (integer): Frequency with which MIP status is printed

By default (zero or negative value) the MIP status is printed after synchronization points. If xslp_mipLog is set to a positive integer, then the current MIP status (node number, best value, best bound) is printed every xslp_mipLog nodes.

Range: \{0, ..., \infty\}

Default: 0

xslp_mipOCount (integer): Number of SLP iterations at each node over which to measure objective function variation

The objective function test for MIP termination is applied only when step bounding has been applied (or xslp_sbStart SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last xslp_mipOCount SLP iterations is within xslp_mipOTol_a or within OBJ * xslp_mipOTol_r where OBJ is the average value of the objective function over those iterations.

Range: \{0, ..., \infty\}

Default: 5

xslp_mipOTol_a (real): Absolute objective function tolerance for MIP termination
The objective function test for MIP termination is applied only when step bounding has been applied (or \texttt{xslp\_sbStart} SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last \texttt{xslp\_mipOCount} SLP iterations is within \texttt{xslp\_mipOTol\_a} or within OBJ \times \texttt{xslp\_mipOTol\_r} where OBJ is the average value of the objective function over those iterations.

Range: \([0, \infty]\)

Default: \(1e^{-05}\)

\texttt{xslp\_mipOTol\_r} (\textit{real}): Relative objective function tolerance for MIP termination

The objective function test for MIP termination is applied only when step bounding has been applied (or \texttt{xslp\_sbStart} SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last \texttt{xslp\_mipOCount} SLP iterations is within \texttt{xslp\_mipOTol\_a} or within OBJ \times \texttt{xslp\_mipOTol\_r} where OBJ is the average value of the objective function over those iterations.

Range: \([0, \infty]\)

Default: \(1e^{-05}\)

\texttt{xslp\_mipRelaxStepBounds} (\textit{integer}): Bitmap describing the step-bound relaxation strategy during MISLP

At any node (including the initial and optimal nodes) it is possible to relax the step bounds of classes of variables so that the variables themselves are completely free to change. This may help with finding a global optimum, but it may also increase the solution time, because more SLP iterations are necessary at each node to obtain a converged solution.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 15

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to \texttt{xslp_mipRelaxStepBounds_structNotCoef}.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to \texttt{xslp_mipRelaxStepBounds_structAll}.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to \texttt{xslp_mipRelaxStepBounds_coefOnly}.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to \texttt{xslp_mipRelaxStepBounds_coef}.</td>
</tr>
</tbody>
</table>

\texttt{xslp\_mipRelaxStepBounds\_coef} (\textit{boolean}): Relax step bounds on SLP variables appearing in coefficients

See also \texttt{xslp\_mipRelaxStepBounds}.

Default: 1

\texttt{xslp\_mipRelaxStepBounds\_coefOnly} (\textit{boolean}): Relax step bounds on SLP variables appearing only in coefficients

See also \texttt{xslp\_mipRelaxStepBounds}.

Default: 1

\texttt{xslp\_mipRelaxStepBounds\_structAll} (\textit{boolean}): Relax step bounds on all structural SLP variables
See also `xslp_mipRelaxStepBounds`.
Default: 1

`xslp_mipRelaxStepBounds_structNotCoeff` (boolean): Relax step bounds on structural SLP variables which are not in coefficients

See also `xslp_mipRelaxStepBounds`.
Default: 1

`xslp_msMaxBoundRange` (real): Defines the maximum range inside which initial points are generated by multistart presets

The is the maximum range in which initial points are generated; the actual range is expected to be smaller as bounds are domains are also considered.

Range: $[0, \infty]$  
Default: 1000

`xslp_mTol_a` (real): Absolute effective matrix element convergence tolerance

The absolute effective matrix element convergence criterion assesses the change in the effect of a coefficient in a constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. $E = X \times C$ where $X$ is the activity of the matrix column in which the coefficient appears, and $C$ is the value of the coefficient. The linearization approximates the effect of the coefficient as $E = X \times C0 + \delta X \times C'0$ where $V$ is as before, $C0$ is the value of the coefficient $C$ calculated using the assumed values for the variables and $C'0$ is the value of $\partial C / \partial X$ calculated using the assumed values for the variables. If $C1$ is the value of the coefficient $C$ calculated using the actual values for the variables, then the error in the effect of the coefficient is given by $\delta E = X \times C1 - (X \times C0 + \delta X \times C'0)$ If $\delta E < X \times xslp_mTol_a$ then the variable has passed the absolute effective matrix element convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: $[-\infty, \infty]$  
Default: auto

`xslp_mTol_r` (real): Relative effective matrix element convergence tolerance

The relative effective matrix element convergence criterion assesses the change in the effect of a coefficient in a constraint relative to the magnitude of the coefficient. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. $E = X \times C$ where $X$ is the activity of the matrix column in which the coefficient appears, and $C$ is the value of the coefficient. The linearization approximates the effect of the coefficient as $E1 = X \times C0 + \delta X \times C'0$ where $V$ is as before, $C0$ is the value of the coefficient $C$ calculated using the assumed values for the variables and $C'0$ is the value of $\partial C / \partial X$ calculated using the assumed values for the variables. If $C1$ is the value of the coefficient $C$ calculated using the actual values for the variables, then the error in the effect of the coefficient is given by $\delta E = X \times C1 - (X \times C0 + \delta X \times C'0)$ If $\delta E < E1 \times xslp_mTol_r$ then the variable has passed the relative effective matrix element convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp_validationTarget_r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: $[-\infty, \infty]$  
Default: auto

`xslp_multistartPreset` (integer): Enable multistart

Default: 0
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable multistart preset.</td>
</tr>
<tr>
<td>1</td>
<td>Generate $\text{xslp_multistart_maxSolves}$ number of random base points.</td>
</tr>
<tr>
<td>2</td>
<td>Generate $\text{xslp_multistart_maxSolves}$ number of random base points, filtered by a merit function centred on initial feasibility.</td>
</tr>
<tr>
<td>3</td>
<td>Load the most typical SLP tuning settings. A maximum of $\text{xslp_multistart_maxSolves}$ jobs are loaded.</td>
</tr>
<tr>
<td>4</td>
<td>Load a comprehensive set of SLP tuning settings. A maximum of $\text{xslp_multistart_maxSolves}$ jobs are loaded.</td>
</tr>
</tbody>
</table>

`xslp\_multistart\_maxSolves` *(integer)*: Maximum number of jobs to create during the multistart search

This control can be increased on the fly during the multistart search: for example, if a job gets refused by a user callback, the callback may increase this limit to account for the rejected job.

Range: $\{-1, \ldots, \infty\}$

Default: unlimited

`xslp\_multistart\_maxTime` *(integer)*: Maximum total time to be spent in the multistart search

XSLP MAXTIME applies on a per job instance basis. There will be some time spent even after $\text{xslp\_multistart\_maxTime}$ has elapsed, while the running jobs get terminated and their results collected.

Range: $\{0, \ldots, \infty\}$

Default: unlimited

`xslp\_multistart\_poolsize` *(integer)*: Maximum number of problem objects allowed to pool up before synchronization in the deterministic multistart

Deterministic multistart is ensured by guaranteeing that the multistart solve results are evaluated in the same order every time. Solves that finish too soon can be pooled until all earlier started solves finish, allowing the system to start solving other multistart instances in the meantime on idle threads. Larger pool sizes will provide better speedups, but will require larger amounts of memory. Positive values are interpreted as a multiplier on the maximum number of active threads used, while negative values are interpreted as an absolute limit (and the absolute value is used). A value of zero will mean no result pooling.

Range: $\{0, \ldots, \infty\}$

Default: 2

`xslp\_multistart\_seed` *(integer)*: Random seed used for the automatic generation of initial point when loading multistart presets

Range: $\{-\infty, \ldots, \infty\}$

Default: 0

`xslp\_multistart\_threads` *(integer)*: Maximum number of threads to be used in multistart


The current hard upper limit on the number of threads to be sued in multistart is 64.

Range: {−1, ..., ∞}

Default: auto

\textbf{xslp\_mvTol} (\textit{real}): Marginal value tolerance for determining if a constraint is slack

If the absolute value of the marginal value of a constraint is less than \text{\texttt{xslp\_mvTol}}, then (1) the constraint is regarded as not constraining for the purposes of the slack tolerance convergence criteria; (2) the constraint is not regarded as an active constraint when identifying unconverged variables in active constraints. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target \text{\texttt{xslp\_validationTarget\_r}}. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)

Default: auto

\textbf{xslp\_objToPenaltyCost} (\textit{real}): Factor to estimate initial penalty costs from objective function

The setting of initial penalty error costs can affect the path of the optimization and, indeed, whether a solution is achieved at all. If the penalty costs are too low, then unbounded solutions may result although Xpress-SLP will increase the costs in an attempt to recover. If the penalty costs are too high, then the requirement to achieve feasibility of the linearized constraints may be too strong to allow the system to explore the nonlinear feasible region. Low penalty costs can result in many SLP iterations, as feasibility of the nonlinear constraints is not achieved until the penalty costs become high enough; high penalty costs force feasibility of the linearizations, and so tend to find local optima close to an initial feasible point. Xpress-SLP can analyze the problem to estimate the size of penalty costs required to avoid an initial unbounded solution. \text{\texttt{xslp\_objToPenaltyCost}} can be used in conjunction with this procedure to scale the costs and give an appropriate initial value for balancing the requirements of feasibility and optimality. Not all models are amenable to the Xpress-SLP analysis. As the analysis is initially concerned with establishing a cost level to avoid unboundedness, a model which is sufficiently constrained will never show unboundedness regardless of the cost. Also, as the analysis is done at the start of the optimization to establish a penalty cost, significant changes in the coefficients, or a high degree of nonlinearity, may invalidate the initial analysis. A setting for \text{\texttt{xslp\_objToPenaltyCost}} of zero disables the analysis. A setting of 3 or 4 has proved successful for many models. If \text{\texttt{xslp\_objToPenaltyCost}} cannot be used because of the problem structure, its effect can still be emulated by some initial experiments to establish the cost required to avoid unboundedness, and then manually applying a suitable factor. If the problem is initially unbounded, then the penalty cost will be increased until either it reaches its maximum or the problem becomes bounded.

Range: \([0, \infty]\)

Default: 0

\textbf{xslp\_oCount} (\textit{integer}): Number of SLP iterations over which to measure objective function variation for static objective (2) convergence criterion

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least \text{\texttt{xslp\_mvTol}}) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as \(\delta\text{Obj} = \text{MAXIter(Obj)} - \text{MINIter(Obj)}\) where \text{Iter} is the \text{\texttt{xslp\_oCount}} most recent SLP iterations and \text{Obj} is the
corresponding objective function value. If $\text{ABS}(\delta \text{Obj}) \leq \text{xslp}_oTol_a$ then the problem has converged on the absolute static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2.

Range: $\{0, \ldots, \infty\}$

Default: 5

**xslp_optimalityTolTarget** *(real)*: When set, this defines a target optimality tolerance to which the linearizations are solved to

This is a soft version of optimalityTol, and will dynamically revert back to optimalityTol if the desired accuracy could not be achieved.

Range: $[0, \infty]$  

Default: not set

**xslp_oTol_a** *(real)*: Absolute static objective (2) convergence tolerance

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least xslp_mvTol) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as $\delta \text{Obj} = \text{MAXIter(Obj)} - \text{MINIter(Obj)}$ where Iter is the xslp_oCount most recent SLP iterations and Obj is the corresponding objective function value. If $\text{ABS}(\delta \text{Obj}) \leq \text{xslp}_oTol_a$ then the problem has converged on the absolute static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: $[-\infty, \infty]$  

Default: auto

**xslp_oTol_r** *(real)*: Relative static objective (2) convergence tolerance

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least xslp_mvtol) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as $\delta \text{Obj} = \text{MAXIter(Obj)} - \text{MINIter(Obj)}$ where Iter is the xslp_oCount most recent SLP iterations and Obj is the corresponding objective function value. If $\text{ABS}(\delta \text{Obj}) \leq \text{xslp}_oTol_r \times \text{xslp}_oTol_a$ then the problem has converged on the relative static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: $[-\infty, \infty]$  

Default: auto

**xslp_penaltyInfoStart** *(integer)*: Iteration from which to record row penalty information
Information about the size (current and total) of active penalties of each row and the number of times a penalty vector has been active is recorded starting at the SLP iteration number given by xslp.penal	
tyInfoStart.

Range: \{0, ..., \infty\}

Default: 3

\texttt{xslp\_postsolve} (boolean): Determines whether postsolving should be performed automatically.

Default: 0
**xslp_presolve** *(integer)*: Determines whether presolving should be performed prior to starting the main algorithm

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not automatically postsolve.</td>
</tr>
<tr>
<td>1</td>
<td>Postsolve automatically.</td>
</tr>
</tbody>
</table>

The Xpress NonLinear nonlinear presolve (which is carried out once, before augmentation) is independent of the Optimizer presolve (which is carried out during each SLP iteration).

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable SLP presolve.</td>
</tr>
<tr>
<td>1</td>
<td>Activate SLP presolve.</td>
</tr>
<tr>
<td>2</td>
<td>Low memory presolve. Original problem is not restored by postsolve and dual solution may not be completely postsolved.</td>
</tr>
</tbody>
</table>

**xslp_presolveLevel** *(integer)*: Determines the level of changes presolve may carry out on the problem

*xslp_presolveOps* controls the operations carried out in presolve. *xslp_presolveLevel* controls how those operations may change the problem.

Default: 4

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Individual rows only presolve, no nonlinear transformations.</td>
</tr>
<tr>
<td>2</td>
<td>Individual rows and bounds only presolve, no nonlinear transformations.</td>
</tr>
<tr>
<td>3</td>
<td>Presolve allowing changing problem dimension, no nonlinear transformations.</td>
</tr>
<tr>
<td>4</td>
<td>Full presolve.</td>
</tr>
</tbody>
</table>

**xslp_presolveOps** *(integer)*: Bitmap indicating the SLP presolve actions to be taken

The Xpress NonLinear nonlinear presolve (which is carried out once, before augmentation) is independent of the Optimizer presolve (which is carried out during each SLP iteration). Linear reductions are performed according to *presolveOps* if bit 12 is not set.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 2104

<table>
<thead>
<tr>
<th>bit value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to <em>xslp_presolveOps_general</em>.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to <em>xslp_presolveOps_fixZero</em>.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to <em>xslp_presolveOps_fixAll</em>.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to <em>xslp_presolveOps_setBounds</em>.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to <em>xslp_presolveOps_intBounds</em>.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to <em>xslp_presolveOps_domain</em>.</td>
</tr>
<tr>
<td>bit 8 = 256</td>
<td>Equivalent to <em>xslp_presolveOps_noCoefficients</em>.</td>
</tr>
<tr>
<td>bit 9 = 512</td>
<td>Equivalent to <em>xslp_presolveOps_noDeltas</em>.</td>
</tr>
<tr>
<td>value</td>
<td>meaning</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>bit 10 = 1024</td>
<td>Equivalent to xslp_presolveOps_noDualSide.</td>
</tr>
<tr>
<td>bit 11 = 2048</td>
<td>Equivalent to xslp_presolveOps_eliminations.</td>
</tr>
<tr>
<td>bit 12 = 4096</td>
<td>Equivalent to xslp_presolveOps_noLinear.</td>
</tr>
<tr>
<td>bit 13 = 8192</td>
<td>Equivalent to xslp_presolveOps_noSimplifier.</td>
</tr>
</tbody>
</table>

**xslp_presolveOps_domain** *(boolean):* Bound tightening based on function domains

See also `xslp_presolveOps`.  
Default: 1

**xslp_presolveOps_eliminations** *(boolean):* Allow eliminations on determined variables

See also `xslp_presolveOps`.  
Default: 1

**xslp_presolveOps_fixAll** *(boolean):* Explicitly fix all columns identified as fixed

See also `xslp_presolveOps`.  
Default: 0

**xslp_presolveOps_fixZero** *(boolean):* Explicitly fix columns identified as fixed to zero

See also `xslp_presolveOps`.  
Default: 0

**xslp_presolveOps_general** *(boolean):* Generic SLP presolve

See also `xslp_presolveOps`.  
Default: 0

**xslp_presolveOps_intBounds** *(boolean):* MISLP bound tightening

See also `xslp_presolveOps`.  
Default: 1

**xslp_presolveOps_noCoefficients** *(boolean):* Do not presolve coefficients

See also `xslp_presolveOps`.  
Default: 0

**xslp_presolveOps_noDeltas** *(boolean):* Do not remove delta variables

See also `xslp_presolveOps`.  
Default: 0

**xslp_presolveOps_noDualSide** *(boolean):* Avoid reductions that can not be dual postsolved
See also `xslp_presolveOps`.

**Default:** 0

**xslp_presolveOps_noLinear (boolean):** Avoid performing linear reductions at the nlp level

See also `xslp_presolveOps`.

**Default:** 0

**xslp_presolveOps_noSimplifier (boolean):** Avoid simplifying nonlinear expressions

See also `xslp_presolveOps`.

**Default:** 0

**xslp_presolveOps_setBounds (boolean):** SLP bound tightening

See also `xslp_presolveOps`.

**Default:** 1

**xslp_presolveZero (real):** Minimum absolute value for a variable which is identified as nonzero during SLP presolve

During the SLP (nonlinear) presolve, a variable may be identified as being nonzero (for example, because it is used as a divisor). A bound of plus or minus `xslp_presolveZero` will be applied to the variable if it is identified as non-negative or non-positive.

**Range:** \([0, \infty]\)

**Default:** \(1e-09\)

**xslp_primalIntegralRef (real):** Reference solution value to take into account when calculating the primal integral

When a global optimum is known, this can be used to calculate a globally valid primal integral. It can also be used to indicate the target objective value still to be taken into account in the integral.

**Range:** \([-\infty, \infty]\)

**Default:** \(1e+20\)

**xslp_probing (integer):** Determines whether probing on a subset of variables should be performed prior to starting the main algorithm

Probing runs multiple times bound reduction in order to further tighten the bounding box.

The Xpress NonLinear nonlinear probing, which is carried out once, is independent of the Optimizer presolve (which is carried out during each SLP iteration). The probing level allows for probing on an expanding set of variables, allowing for probing on all variables (level 5) or only those for which probing is more likely to be useful (binary variables).

**Default:** auto
<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic.</td>
</tr>
<tr>
<td>0</td>
<td>Disable SLP probing.</td>
</tr>
<tr>
<td>1</td>
<td>Activate SLP probing only on binary variables.</td>
</tr>
<tr>
<td>2</td>
<td>Activate SLP probing only on binary or unbounded integer variables.</td>
</tr>
<tr>
<td>3</td>
<td>Activate SLP probing only on binary or integer variables.</td>
</tr>
<tr>
<td>4</td>
<td>Activate SLP probing only on binary, integer variables, and unbounded continuous variables.</td>
</tr>
<tr>
<td>5</td>
<td>Activate SLP probing on any variable.</td>
</tr>
</tbody>
</table>

**xslp_sameCount (integer):** Number of steps reaching the step bound in the same direction before step bounds are increased $\leftrightarrow$

If step bounding is enabled, the step bound for a variable will be increased if successive changes are in the same direction. More precisely, if there are $xslp\_sameCount$ successive changes reaching the step bound and in the same direction for a variable, then the step bound (B) for the variable will be reset to $B \times xslp\_expand$.

Range: $\{0, \ldots, \infty\}$

Default: 3

**xslp_sameDamp (integer):** Number of steps in same direction before damping factor is increased $\leftrightarrow$

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are $xslp\_sameDamp$ successive changes in the same direction for a variable, then the damping factor (D) for the variable will be reset to $D \times xslp\_dampExpand + xslp\_dampMax \times (1 - xslp\_dampExpand)$

Range: $\{0, \ldots, \infty\}$

Default: 3

**xslp_sbStart (integer):** SLP iteration after which step bounds are first applied $\leftrightarrow$

If step bounds are used, they can be applied for the whole of the SLP optimization process, or started after a number of SLP iterations. In general, it is better not to apply step bounds from the start unless one of the following applies: (1) the initial estimates are known to be good, and explicit values can be provided for initial step bounds on all variables; or (2) the problem is unbounded unless all variables are step-bounded.

Range: $\{0, \ldots, \infty\}$

Default: 8

**xslp_scale (integer):** When to re-scale the SLP problem $\leftrightarrow$

During the SLP optimization, matrix entries can change considerably in magnitude, even when the formulae in the coefficients are not very nonlinear. Re-scaling of the matrix can reduce numerical errors, but may increase the time taken to achieve convergence.

Default: 1

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No re-scaling.</td>
</tr>
<tr>
<td>1</td>
<td>Re-scale every SLP iteration up to $xslp_scaleCount$ iterations after the end of barrier optimization.</td>
</tr>
<tr>
<td>2</td>
<td>Re-scale every SLP iteration up to $xslp_scaleCount$ iterations in total.</td>
</tr>
<tr>
<td>3</td>
<td>Re-scale every SLP iteration until primal simplex is automatically invoked.</td>
</tr>
<tr>
<td>4</td>
<td>Re-scale every SLP iteration.</td>
</tr>
</tbody>
</table>
**xslp_scaleCount** (integer): Iteration limit used in determining when to re-scale the SLP matrix

If **xslp_scale** is set to 1 or 2, then **xslp_scaleCount** determines the number of iterations (after the end of barrier optimization or in total) in which the matrix is automatically re-scaled.

Range: \{0, ..., \infty\}

Default: 0

**xslp_shrink** (real): Multiplier to reduce a step bound

If step bounding is enabled, the step bound for a variable will be decreased if successive changes are in opposite directions. The step bound (B) for the variable will be reset to B \times **xslp_shrink**.

If the step bound is already below the strict (delta or closure) tolerances, it will not be reduced further.

Range: [0, 1]

Default: 0.5

**xslp_shrinkBias** (real): Defines an overwrite / adjustment of step bounds for improving iterations

Positive values overwrite **xslp_shrink** only if the objective is improving. A negative value is used to scale all step bounds in improving iterations.

Range: [-\infty, \infty]

Default: not set

**xslp_slpLog** (integer): Frequency with which SLP status is printed

If **xslp_log** is set to zero (minimal logging) then a nonzero value for **xslp_slpLog** defines the frequency (in SLP iterations) when summary information is printed out.

Range: \{0, ..., \infty\}

Default: 1

**xslp_solver** (integer): First order differentiation mode when using analytical derivatives

The presence of KNITRO is detected automatically. KNITRO can be used to solve any problem loaded into XSLP, independently from how the problem was loaded. **xslp_solver** is set to automatic, Xpress SLP will be selected if any SLP specific construct has been loaded (these are ignored if KNITRO is selected manually).

Default: auto

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Automatic selection, based on model characteristics and solver availability</td>
</tr>
<tr>
<td>0</td>
<td>Use Xpress-SLP (always available)</td>
</tr>
<tr>
<td>1</td>
<td>Use Knitro if available</td>
</tr>
</tbody>
</table>

**xslp_sTol_a** (real): Absolute slack convergence tolerance

The slack convergence criterion is identical to the impact convergence criterion, except that the tolerances used are **xslp_sTol_a** (instead of **xslp_iTol_a**) and **xslp_sTol_r** (instead of **xslp_iTol_r**).
See `xslp iTol a` for a description of the test. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp validationTarget r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\]

Default: `auto`

**xslp sTol r (real):** Relative slack convergence tolerance

The slack convergence criterion is identical to the impact convergence criterion, except that the tolerances used are `xslp sTol a` (instead of `xslp iTol a`) and `xslp sTol r` (instead of `xslp iTol r`). See `xslp iTol r` for a description of the test. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target `xslp validationTarget r`. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\]

Default: `auto`

**xslp_stopOutOfRange (boolean):** Stop optimization and return error code if internal function argument is out of range

If `xslp_stopOutOfRange` is set to 1, then if an internal function receives an argument which is out of its allowable range (for example, LOG of a negative number), an error code is set and the optimization is terminated.

Default: 0

**xslp_threads (integer):** Default number of threads to be used

Overall thread control value, used to determine the number of threads used where parallel calculations are possible.

Range: \([-1, ..., \infty]\]

Default: `auto`

**xslp_unfinishedLimit (integer):** Number of consecutive SLP iterations that may have an unfinished status before the solve is terminated

If the optimization of the current linear approximation terminates with an "unfinished" status, then first a number of strategies are applied to attempt a successful solve of the same linearization. If this fails, then a new iteration is started to change the linearization itself. This control limits the number of such repeated attempts.

Range: \([0, ..., \infty]\]

Default: 3

**xslp validationTarget k (real):** Optimality target tolerance

Primary optimality control for SLP. When the relevant optimality based convergence controls are left at their default values, SLP will adjust their value to match the target. The control defines a target value, that may not necessarily be attainable for problem with no strong constraint qualifications.

Range: \([0, \infty]\]

Default: `1e-06`
**xslp.validationTarget_r (real):** Feasibility target tolerance

Primary feasibility control for SLP. When the relevant feasibility based convergence controls are left at their default values, SLP will adjust their value to match the target. The control defines a target value, that may not necessarily be attainable.

Range: \([0, \infty]\)

Default: \(1e-06\)

**xslp.vCount (integer):** Number of SLP iterations over which to measure static objective (3) convergence

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as \(\delta\text{Obj} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj})\) where \(\text{Iter}\) is the xslp.vCount most recent SLP iterations and \(\text{Obj}\) is the corresponding objective function value. If \(\text{ABS}(\delta\text{Obj}) \leq \text{xslp.vTol.a}\) then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp.vLimit + xslp.sbStart SLP iterations have taken place and only if xslp.vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced.

Range: \(\{0, ..., \infty\}\)

Default: 0

**xslp.vLimit (integer):** Number of SLP iterations after which static objective (3) convergence testing starts

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as \(\delta\text{Obj} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj})\) where \(\text{Iter}\) is the xslp.vCount most recent SLP iterations and \(\text{Obj}\) is the corresponding objective function value. If \(\text{ABS}(\delta\text{Obj}) \leq \text{xslp.vTol.a}\) then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp.vLimit + xslp.sbStart SLP iterations have taken place and only if xslp.vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced.

Range: \(\{0, ..., \infty\}\)

Default: 0

**xslp.vTol.a (real):** Absolute static objective (3) convergence tolerance

---
The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as \( \delta \text{Obj} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj}) \) where Iter is the xspr.vCount most recent SLP iterations and Obj is the corresponding objective function value. If \( \text{ABS}(\delta \text{Obj}) \leq \text{xslp.vTol.a} \) then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp.vLimit + xslp.sbStart SLP iterations have taken place and only if xslp.vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp.validationTarget.k. Good values for the control are usually fall between 1e-3 and 1e-6.

\[
\delta \text{Obj} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj})
\]

Range: \([-\infty, \infty]\)

Default: auto

**xslp.vTol.r (real):** Relative static objective (3) convergence tolerance

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as \( \delta \text{Obj} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj}) \) where Iter is the xspr.vCount most recent SLP iterations and Obj is the corresponding objective function value. If \( \text{ABS}(\delta \text{Obj}) \leq \text{AVGIter}(\text{Obj}) \times \text{xslp.vTol.t} \) then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp.vLimit.

- xslp.sbStart SLP iterations have taken place and only if xslp.vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp.validationTarget.k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)

Default: auto

**xslp.wCount (integer):** Number of SLP iterations over which to measure the objective for the extended convergence continuation criterion

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the last converged solution is given by: \( \delta \text{Obj} = \text{Obj} - \text{LastConvergedObj} \) For a minimization problem, the sign is reversed. If \( \delta \text{Obj} > \text{xslp.wTol.a} \) and \( \delta \text{Obj} > \text{ABS}(\text{ConvergedObj}) \times \text{xslp.wTol.r} \)
then the solution is deemed to have a significantly better objective function value than the converged solution. When a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the \texttt{xslp\_wCount} most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. If \texttt{xslp\_wCount} is zero, then the extended convergence continuation criterion is disabled.

Range: \{0, \ldots, \infty\}

Default: 0

\texttt{xslp\_wTol\_a} (\textit{real}): Absolute extended convergence continuation tolerance

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the last converged solution is given by: \( \delta \text{Obj} = \text{Obj} - \text{LastConvergedObj} \). For a minimization problem, the sign is reversed. If \( \delta \text{Obj} > \texttt{xslp\_wTol\_a} \) and \( \delta \text{Obj} > \text{ABS(ConvergedObj)} \times \texttt{xslp\_wTol\_r} \), then the solution is deemed to have a significantly better objective function value than the converged solution. When a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the \texttt{xslp\_wCount} most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target \texttt{xslp\_validationTarget\_k}. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: \([-\infty, \infty]\)

Default: \texttt{auto}

\texttt{xslp\_wTol\_r} (\textit{real}): Relative extended convergence continuation tolerance

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the
last converged solution is given by: \( \delta \text{Obj} = \text{Obj} - \text{LastConvergedObj} \). For a minimization problem, the sign is reversed. If \( \delta \text{Obj} > xslp_wTol_a \) and \( \delta \text{Obj} > \text{ABS(ConvergedObj)} \times xslp_wTol_r \) then the solution is deemed to have a significantly better objective function value than the converged solution. If \( xslp_wCount \) is greater than zero, and a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the \( xslp_wCount \) most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target \( xslp_validationTarget_k \). Good values for the control are usually fall between 1e-4 and 1e-6.

Range: \([-\infty, \infty]\)

Default: auto

**xslp_xCount (integer):** Number of SLP iterations over which to measure static objective (1) convergence

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as \( \delta \text{Obj} = \text{MAXIter(Obj)} - \text{MINIter(Obj)} \) where Iter is the \( xslp_xCount \) most recent SLP iterations and Obj is the corresponding objective function value. If \( \text{ABS}(\delta \text{Obj}) \leq xslp_xTol_a \) then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If \( \text{ABS}(\delta \text{Obj}) \leq \text{AVGIter(Obj)} \times xslp_xTol_r \) then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until \( xslp_xLimit \) SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged.

Range: \( \{0, ..., \infty\} \)

Default: 5

**xslp_xLimit (integer):** Number of SLP iterations up to which static objective (1) convergence testing starts

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The
The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as $\delta_{\text{Obj}} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj})$ where Iter is the $\text{xslp}_x\text{Count}$ most recent SLP iterations and Obj is the corresponding objective function value. If $\text{ABS}(\delta_{\text{Obj}}) \leq \text{xslp}_x\text{Tol}_a$ then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If $\text{ABS}(\delta_{\text{Obj}}) \leq \text{AVGIter}(\text{Obj}) \times \text{xslp}_x\text{Tol}_r$ then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until $\text{xslp}_x\text{Limit}$ SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged.

Range: $\{0, \ldots, \infty\}$

Default: 100

$\text{xslp}_x\text{Tol}_a$ (real): Absolute static objective function (1) tolerance

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as $\delta_{\text{Obj}} = \text{MAXIter}(\text{Obj}) - \text{MINIter}(\text{Obj})$ where Iter is the $\text{xslp}_x\text{Count}$ most recent SLP iterations and Obj is the corresponding objective function value. If $\text{ABS}(\delta_{\text{Obj}}) \leq \text{xslp}_x\text{Tol}_a$ then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If $\text{ABS}(\delta_{\text{Obj}}) \leq \text{AVGIter}(\text{Obj}) \times \text{xslp}_x\text{Tol}_r$ then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until $\text{xslp}_x\text{Limit}$ SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target $\text{xslp}_x\text{ValidationTarget}_k$.

Good values for the control are usually fall between $1e^{-3}$ and $1e^{-6}$.

Range: $[-\infty, \infty]$

Default: auto

$\text{xslp}_x\text{Tol}_r$ (real): Relative static objective function (1) tolerance

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then
there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as $\delta_{Obj} = \maxIter(Obj) - \minIter(Obj)$ where $\text{Iter}$ is the $xslp_xCount$ most recent SLP iterations and $\text{Obj}$ is the corresponding objective function value. If $\text{ABS}(\delta_{Obj}) \leq xslp_xTol_a$ then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If $\text{ABS}(\delta_{Obj}) \leq \text{AVGIter}(Obj) \times xslp_xTol_r$ then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until $xslp_xLimit$ SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target $xslp_validationTarget_k$. Good values for the control are usually fall between 1e-4 and 1e-6.

Range: $[-\infty, \infty]$

Default: auto

$xslp_zero$ (real): Absolute tolerance ←

If a value is below $xslp_zero$ in magnitude, then it will be regarded as zero in certain formula calculations: an attempt to divide by such a value will give a "divide by zero" error; an exponent of a negative number will produce a "negative number, fractional exponent" error if the exponent differs from an integer by more than $xslp_zero$.

Range: $[0, \infty]$

Default: 1e-15

$xslp_zeroCriterion$ (integer): Bitmap determining the behavior of the placeholder deletion procedure ←

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 = 1</td>
<td>Equivalent to $xslp_zeroCriterion_nbSLPVar$.</td>
</tr>
<tr>
<td>bit 1 = 2</td>
<td>Equivalent to $xslp_zeroCriterion_nbDelta$.</td>
</tr>
<tr>
<td>bit 2 = 4</td>
<td>Equivalent to $xslp_zeroCriterion_slpVarNBUpdateRow$.</td>
</tr>
<tr>
<td>bit 3 = 8</td>
<td>Equivalent to $xslp_zeroCriterion_deltaNBUpdateRow$.</td>
</tr>
<tr>
<td>bit 4 = 16</td>
<td>Equivalent to $xslp_zeroCriterion_deltaNBDRRow$.</td>
</tr>
<tr>
<td>bit 5 = 32</td>
<td>Equivalent to $xslp_zeroCriterion_print$.</td>
</tr>
</tbody>
</table>

$xslp_zeroCriterionCount$ (integer): Number of consecutive times a placeholder entry is zero before being considered for deletion ←

Range: $\{0, ..., \infty\}$

Default: 0
**xslp_zeroCriterionStart** *(integer)*: SLP iteration at which criteria for deletion of placeholder entries are first activated

Range: \{0, ..., ∞\}

Default: 0

**xslp_zeroCriterion_deltaNBDRRow** *(boolean)*: Remove placeholders in a basic delta variable if the determining row for the corresponding SLP variable is nonbasic

See also **xslp_zeroCriterion**.

Default: 0

**xslp_zeroCriterion_deltaNBUpdateRow** *(boolean)*: Remove placeholders in a basic delta variable if its update row is nonbasic and the corresponding SLP variable is nonbasic

See also **xslp_zeroCriterion**.

Default: 0

**xslp_zeroCriterion_nbDelta** *(boolean)*: Remove placeholders in nonbasic delta variables

See also **xslp_zeroCriterion**.

Default: 0

**xslp_zeroCriterion_nbSLPVar** *(boolean)*: Remove placeholders in nonbasic SLP variables

See also **xslp_zeroCriterion**.

Default: 0

**xslp_zeroCriterion_print** *(boolean)*: Print information about zero placeholders

See also **xslp_zeroCriterion**.

Default: 0

**xslp_zeroCriterion_slpVarNBUpdateRow** *(boolean)*: Remove placeholders in a basic SLP variable if its update row is nonbasic

See also **xslp_zeroCriterion**.

Default: 0
5.42.5 Helpful Hints

The comments below should help both novice and experienced GAMS users to better understand and make use of GAMS/XPRESS.

- **Infeasible and unbounded models** The fact that a model is infeasible/unbounded can be detected at two stages: during the presolve and during the simplex or barrier algorithm. In the first case we cannot recover a solution, nor is any information regarding the infeasible/unbounded constraint or variable provided (at least in a way that can be returned to GAMS). In such a situation, the GAMS link will automatically rerun the model using primal simplex with presolve turned off (this can be avoided by setting the rerun option to 0). It is possible (but very unlikely) that the simplex method will solve a model to optimality while the presolve claims the model is infeasible/unbounded (due to feasibility tolerances in the simplex and barrier algorithms).

- The barrier method does not make use of `iterlim`. Use `bariterlim` in an options file instead. The number of barrier iterations is echoed to the log and listing file. If the barrier iteration limit is reached during the barrier algorithm, XPRESS continues with a simplex algorithm, which will obey the `iterlim` setting.

- Semi-integer variables are not implemented in the link, nor are they supported by XPRESS; if present, they trigger an error message.

- SOS1 and SOS2 variables are required by XPRESS to have lower bounds of 0 and nonnegative upper bounds.

5.42.6 Setting up a GAMS/XPRESS-Link license

To use the GAMS/XPRESS solver with a GAMS/XPRESS-Link license you have to set up the XPRESS portion of the licensing. To do so, copy your XPRESS license `xpauth.xpr` to the GAMS system directory. As of version 24.2 GAMS already comes with a file `xpauth.xpr`. You might consider copying this file to `xpauth.xpr.bak` or similar before overwriting it with your own XPRESS license file.
Chapter 6

Tools Manuals

A large number of tools are included in GAMS distribution. Below are a functional categorization of all tools, an alphabetically sorted list of all tools, and a brief description of each tool with their supported platforms.

6.1 Tools Category

All tools included in GAMS distribution are categorized as

6.1.1 GAMS Integrated Development Environments

There are two integrated model development environments including a general text editor with the ability to launch and monitor the compilation and execution of GAMS models: The GAMS IDE and GAMS Studio.

6.1.2 Data Exchange

A collection of tools that provide functionality to exchange data between GAMS and other data sources. This category contains tools for popular data sources and high-level programming environment and like databases (GDX2ACCESS, GDX2SQLITE, MDB2GMS, SQL2GMS), Matlab (GDXMRW), and R (GDXRRW). There are also tools for specialized systems like HAR (GDX2HAR and HAR2GDX), TROLL (GDXTROLL), and VEDA (GDX2VEDA).

6.1.2.1 Excel

Tools to exchange data with Microsoft Excel are grouped in a subcategory Excel. The tools in this category are GDX2XLS, GDXXRW, MSAPPAVAIL, XLS2GMS, XLSDUMP, and XLSTALK. Many of the tools described here use the GAMS Data eXchange facility GAMS Data eXchange (GDX).

6.1.3 GDX Service

A collection of tools that operate directly on GAMS Data eXchange (GDX) containers to e.g. compare (GDXDIFF), copy (GDXCOPY), merge (GDXMERGE), and rename (GDXRENAME).
6.1.4 Data Transformation

A collection of tools that perform very specific tasks that are awkward or inefficient to implement in GAMS directly. Through GAMS Data eXchange (GDX) and the execution of the tools in this category allow to perform complex tasks from a GAMS model like identifying eigenvalues or the inverse of a matrix.

The tools in this category are CHOLESKY, CSDP, EIGENVECTOR, EIGENVALUE, GDXRANK, MessageReceiverWindow, INVERT, MCFILTER, SCENRED, and SCENRED2.

6.1.5 Other Tools

A collection of more exotic tools that can become handy in some some special circumstances. The tools in this category are ASK, CHK4UPD, ENDECRYPT, FINDTHISGAMS, GAMS Posix Utilities, MODEL2TEX, and ShellExecute. Most notably, the collection contains the tool MODEL2TEX to document the model algebra in LaTeX format.

6.2 List of Tools

The following table gives an alphabetically sorted list of all available tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK</td>
<td>The utility can be used to get input from an user interactively.</td>
</tr>
<tr>
<td>CHK4UPD</td>
<td>Checks whether the user can update to a more recent GAMS version.</td>
</tr>
<tr>
<td>CHOLESKY</td>
<td>Calculates the Cholesky decomposition of a symmetric positive definite matrix.</td>
</tr>
<tr>
<td>CSDP</td>
<td>The semidefinite programming CSDP solver from COIN-OR. The communication with CSDP requires the setup of matrix data structures in a CSDP input file. In a sense a GAMS model functions as a matrix generator.</td>
</tr>
<tr>
<td>CSV2GDX</td>
<td>Reads a CSV file (comma separated values) and writes to a GDX file.</td>
</tr>
<tr>
<td>EIGENVALUE</td>
<td>Calculates eigenvalues of a symmetric matrix.</td>
</tr>
<tr>
<td>EIGENVECTOR</td>
<td>Calculates eigenvector of a symmetric matrix.</td>
</tr>
<tr>
<td>ENDECRYPT</td>
<td>A tool to encrypt and decrypt text files.</td>
</tr>
<tr>
<td>FINDTHISGAMS</td>
<td>Windows command line tool for modifying GAMS specific registry entries created by the GAMS installer.</td>
</tr>
<tr>
<td>GAMS IDE</td>
<td>Classic Integrated Development Environment.</td>
</tr>
<tr>
<td>GAMS STUDIO</td>
<td>Integrated Development Environment.</td>
</tr>
<tr>
<td>GDX2ACCESS</td>
<td>Converts GDX data to MS Access tables.</td>
</tr>
<tr>
<td>GDX2HAR</td>
<td>Translates files between GDX and HAR format.</td>
</tr>
<tr>
<td>GDX2SQLITE</td>
<td>Dumps the complete contents of a GDX file into a SQLite2 database. From Amsterdam Optimization Modeling Group.</td>
</tr>
<tr>
<td>GDX2VEDA</td>
<td>Translates a GDX file into the VEDA format.</td>
</tr>
<tr>
<td>GDX2XLS</td>
<td>Converts GDX data into a MS Excel spreadsheet.</td>
</tr>
<tr>
<td>GDXCOPY</td>
<td>Converts a GDX file into different GDX formats.</td>
</tr>
<tr>
<td>GDXDIFF</td>
<td>Compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file.</td>
</tr>
<tr>
<td>Tool</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GDXDUMP</td>
<td>Writes scalars, sets and parameters (tables) to standard output formatted as a GAMS program with data statements.</td>
</tr>
<tr>
<td>GDXMERGE</td>
<td>Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.</td>
</tr>
<tr>
<td>GDXMRW</td>
<td>A suite of utilities to import/export data between GAMS and MATLAB and to call GAMS models from MATLAB and get results back into MATLAB.</td>
</tr>
<tr>
<td>GDXRANK</td>
<td>Reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.</td>
</tr>
<tr>
<td>GDXRENAME</td>
<td>Replaces UEL strings in GDX files.</td>
</tr>
<tr>
<td>GDXRRW</td>
<td>An interface between GAMS and R. It includes functions to transfer data between GDX and R and a function to call GAMS from R.</td>
</tr>
<tr>
<td>GDXTROLL</td>
<td>Translates a GDX file into the TROLL format.</td>
</tr>
<tr>
<td>GDVVIEWER</td>
<td>Views and converts data contained in GDX files.</td>
</tr>
<tr>
<td>GDXXRW</td>
<td>Preferred utility to read and write MS Excel spreadsheet data.</td>
</tr>
<tr>
<td>GMSUNZIP</td>
<td>A copy of the decompression tool unzip from Info-ZIP prefixed with &quot;gms&quot; for clear identification.</td>
</tr>
<tr>
<td>GMSZIP</td>
<td>A copy of the compression and archiving tool zip from Info-ZIP prefixed with &quot;gms&quot; for clear identification.</td>
</tr>
<tr>
<td>HAR2GDX</td>
<td>Translates files between GDX and HAR format.</td>
</tr>
<tr>
<td>IDECMS</td>
<td>Sends commands to the GAMSIDE.</td>
</tr>
<tr>
<td>INVERT</td>
<td>Inverts a matrix.</td>
</tr>
<tr>
<td>MCFILTER</td>
<td>Removal of duplicate and dominated points in a multi-criteria solution set.</td>
</tr>
<tr>
<td>MDB2GMS</td>
<td>Converts data from an MS Access database into a GAMS readable format.</td>
</tr>
<tr>
<td>MESSAGE RECEIVER WINDOW</td>
<td>A graphical tool that receives and displays Windows messages.</td>
</tr>
<tr>
<td>MODEL2TEX</td>
<td>Translates a GAMS model into LaTeX.</td>
</tr>
<tr>
<td>MPS2GMS</td>
<td>Translates an MPS or LP file into an equivalent short generic GAMS program using a GDX file to store data.</td>
</tr>
<tr>
<td>MSAPPAVAIL</td>
<td>Checks if a MS Office Application is available.</td>
</tr>
<tr>
<td>POSIX</td>
<td>A collection of POSIX utilities which are usually available for Windows and the different Unix systems and therefore help to write platform independent scripts.</td>
</tr>
<tr>
<td>SCENRED</td>
<td>A tool for the reduction of scenarios that model random data processes of a stochastic program. From Humboldt-University Berlin.</td>
</tr>
<tr>
<td>SCENRED2</td>
<td>Scenred2 is a fundamental update of Scenred and offers a scenario tree construction algorithm. From Humboldt-University Berlin.</td>
</tr>
<tr>
<td>SHELLEXECUTE</td>
<td>Launches external programs from the command line.</td>
</tr>
<tr>
<td>SQL2GMS</td>
<td>Converts data from an SQL database into a GAMS readable format.</td>
</tr>
<tr>
<td>XLS2GMS</td>
<td>Converts spreadsheet data from a MS Excel spreadsheet into a GAMS readable format.</td>
</tr>
<tr>
<td>XLSDUMP</td>
<td>Writes all worksheets of a MS Excel workbook to a GDX file. Unlike gdxxrw, the program does not require that Excel is installed.</td>
</tr>
<tr>
<td>XLSTALK</td>
<td>Open/Close/Run macro in MS Excel.</td>
</tr>
</tbody>
</table>
6.3 Supported Platforms

<table>
<thead>
<tr>
<th>Tool</th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASK</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHK4UPD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHOLESKY</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CSDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CSV2GDX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>EIGENVALUE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>EIGENVECTOR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ENDECRYPT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FINDTHISGAMS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMSIDE</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMSSTUDIO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDX2ACCESS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDX2HAR</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDX2SQLITE</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GDX2VEDA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDX2XLS</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GDXCOPY</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXDIFF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXDUMP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXMERGE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXMRW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXRANK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXRENAME</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXRRW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXTROLL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GDXVIEWER</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDXXRW</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMSUNZIP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GMSZIP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HAR2GDX</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDECMDS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVERT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MCFILTER</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDB2GMS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSGRWIN</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODEL2TEX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MPS2GMS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MSAPPAVAIL</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSIX**</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCENRED</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCENRED2</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHELLEXECUTE</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL2GMS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLS2GMS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLSDUMP</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLSTALK</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note that the tool is 32 bit but runs fine on 64 bit Windows.

awk, cat, cksum, cmp, comm, cp, cut, diff, expr, fold, gdate, grep, gsort, gunzip, gzip, head, join, make, mkdir, mv, od, paste, printf, rm, sed, sleep, tail, tar, tee, test, touch, tr, uniq, wc, xargs

6.4 ASK

Writing a GUI (Graphical User Interface) for a GAMS application requires some serious programming, and therefore place a burden on the knowledge and time of the modeler. In this section we show how extremely simple user interfaces can be built using a few simple tools. The main purpose of these tools is to allow a developer quickly put an application together such that an end-user does not have to edit GAMS files. We assume the end-user runs a GAMS model from the GAMS-IDE.

The ASK utility can be used to get input from an end-user and the GDXViewer tool can be used to present end-results. Together these tools allow you to build a minimalist GUI without any programming.

6.4.1 Usage

ask <options>

where the options are

T=string

where the string identifies the type of input item to go after and can be

- integer - when one wants an integer number
- float - when one wants a real number
- radiobutton - when one wants a radio button choice
- combobox - when one wants a combo (drop down choice) box
- checklistbox - when one wants a check list box
- fileopenbox - when one wants the name of a file to open
- filesavebox - when one wants the name of a file to save

Example: T=integer

M="string"

where the string is the text to in the box.
Example: M="Enter a number"

O="filename"

where the filename is the name of a file in which to place the results for subsequent inclusion into GAMS
Example: O="file.inc"

D="string 1|string 2..."

where the "string 1|string 2|string 3|...|string n" gives the n strings to be associated with multiple choices when using checkbox, radiobutton, combobox, or checklistbox. The individual strings are separated by the delimiter "|"
Example: D="Small data set|Medium data set|Large data set"
\[ E = \text{"number 1|number 2..."} \]

where the "number 1|number 2|number 3|...|number n" gives the n numbers to be returned to GAMS associated with the choices made when using checkbox, radiobutton, combobox, or checklistbox. The individual numbers are separated by the delimiter "|".
Example: \[ E = \text{"1|2|3|4|5"} \]

\[ I = \text{"filepath"} \]

where filepath gives the path in which to look for the file under the fileopenbox and filesavebox dialogues. If not specified this is the project directory.
Example: \[ I = \text{"C:\gams\mine"} \]

\[ F = \text{"filenmask"} \]

where filenmask gives the mask for acceptable files under the fileopenbox and filesavebox dialogues. If not specified this is *.*.
Example: \[ I = \text{"*.gdx"} \]

\[ R = \text{"string"} \]

where the string gives a line of GAMS code to place in the include file. This can contain a s parameter in which the information to return is substituted.
Example: \[ R = \text{"\$include \"%s\"" or R = \"set i /1990+%s/;\"} \]

\[ C = \text{"string"} \]

A title for the dialogue box being used.
Example: \[ C = \text{"Box to ask for a file"} \]

\[ L = \text{number} \]

where the number gives a lower bound on a numeric entry.
Example: \[ L = 15 \]

\[ U = \text{number} \]

where the number gives an upper bound on a numeric entry.
Example: \[ U = 15 \]

\[ @ = \text{"filename"} \]

where filename gives the name of a file of input instructions containing the options above in this table.
Example: \[ @ = \text{ask.opt} \]

In addition a number by itself can be entered to put multiple entries into columns under the checkbox, radiobutton, combobox, or checklistbox entries.
6.4.2 Calling ASK utility from GAMS

The ASK utility is a simple tool to ask simple interactive questions to the end-user. For instance, if your model requires a scalar to be changed regularly, instead of letting the end-user change the .gms source file, it may be better to pop up a window, with a question text, where the required number can be entered. The ASK tool allows you to do this. As the ASK tool generates a standard GAMS include file, this file can then be used through a $include statement:

```gams
$call =ask T=integer M="Enter number of cities" o=n.inc
Scalar n 'number of cities' /
$include n.inc
display n;
```

The $call statement will invoke the ASK tool. If ASK.EXE is not located in the GAMS system directory but placed somewhere else you may have to provide a path, as in:

```gams
$call =d:\util\ask T=integer M="Enter number of cities" o=n.inc
```

The parameter T=integer indicates we want to ask for an integer number (T means type). The parameter M="text" specifies the question text. Finally O=filename sets the name of the include file that ASK should create.

When we run this GAMS fragment from the IDE, we will see:

![Image of the active process window showing the ASK utility call and the window for entering the number of cities]
After enter a number and hitting the OK button GAMS will continue. The listing file will demonstrate clearly how the include file was formed:

```gams
2 scalar n 'number of cities' / 
INCLUDE D:\GAMS PROJECTS\ASK\N.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 12
7 /;
8 display n;
Include File Summary
SEQ GLOBAL TYPE PARENT LOCAL FILENAME
1 1 INPUT 0 0 D:\GAMS PROJECTS\ASK\X.GMS
2 1 CALL 1 1 =ask T=integer M="Enter number of cities" o=n.inc
3 3 INCLUDE 1 3 .D:\GAMS PROJECTS\ASK\N.INC
```

In this case 12 was entered and the OK button was pressed. If the user pressed the CANCEL button, an error will be generated and the listing file will show that the CANCEL button was pressed:

```gams
2 scalar n 'number of cities' / 
INCLUDE D:\GAMS PROJECTS\ASK\N.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 * Cancel was pressed
7 /;
* **** $1
* **** LINE 4 INPUT D:\GAMS PROJECTS\ASK\X.GMS
* **** 1 Real number expected
8 display n;
```

In case you want to limit the integer that the user is allowed to enter you can specify a lower and an upperbound as in:

```gams
$call =ask T=integer M="Give integer, between 0 and 5" L=0 U=5 O=n1.inc
Scalar n1 'an integer 0..5' /
$include n1.inc
/;
display n1;
```

This will only accept values between 0 and 5.

To allow the user to specify a floating point number, we can use T=float. An example is:

```gams
$call =ask T=float M="Give floating point number, no bounds" O=x.inc
Scalar x 'real' /
$include x.inc
/;
display x;
```

The floating point popup window can be told to make sure the number entered is within certain bounds, using the L=lowerbound and U=upperbound syntax:
$call = ask T=float M="Give floating point number, between 0 and 5" L=0 U=5.0 O=x1.inc
Scalar x1 'real' /
$include x1.inc
/;
display x1;

Up to now, the include file generated by ASK just contained a single number. ASK can generate more complicated include files. Consider the example:

* import a set
$call = ask T=integer M="Give year between 1990 and 2010." C="My Title" L=1990 U=2010 R="set i /1990*%s/;" O=i.inc
$include i.inc
display i;

The parameters C="caption" and R="resultstring" are new. The caption is simple: it sets the name of the window. The result string is a string that is returned after ASK has substituted \( s \) for the result. I.e. if we enter 1991, then the result written to the include file is set i /1990*1991/;

The listing file can be used to check the correct behavior:

1 * import a set
INCLUDE D:\GAMS PROJECTS\ASK\I.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 set i /1990*1991/;
Radio buttons can be used through the parameter T=radiobuttons as in:

* import a number through radio buttons
$call =ask T=radiobuttons M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" E="1|2|3|4|5" R="scalar n2 option /%s/;" O=n2.inc
$include n2.inc
display n2;

The parameter D="option 1|option 2|option 3|option 4|option 5" specifies the text of the options shown. The list E="1|2|3|4|5" gives the return (exit) strings when a certain option is chosen. I.e. if the second option os chosen in the list specified with the D parameter, then the second string in the E list is returned. The result is again substituted in the string specified in the R parameter if it exists.

In this example the command line became rather long and difficult to handle. In addition some Windows systems have restrictive maximum lengths for the command line. Therefore we offer the possibility to specify the command line arguments in a separate external text file. This text file is passed using

@filename

. Assume the file ask.opt looks like:

T=radiobuttons
M=Choose single option
D=option 1|option 2|option 3|option 4|option 5|option 6|option 7|option 8|option 9|option 10
E=1|2|3|4|5|6|7|8|9|10
2
R=scalar n3 option /%s/;
O=n3.inc
6.4 ASK

2663

Every command line parameter is specified on a separate line. Notice the strange option 2; this tells ASK
to display the radio buttons in two columns. We can use this option file as follows:
* id, now 2 columns and using a parameter file
$call =ask @"ask.opt"
$include n3.inc
display n3;
The quotes around the filename are optional, and really only needed if the filename contains blanks.

If you want to keep all the logic in one place, then one can use GAMS to generate the option file. It is
noted that it is not possible to use the PUT facility for this. I.e.:
File f /asktest.opt/;
put f;
put ’T=checklistbox’/
’M=Choose multiple options’/
’D=option 1|option 2|option 3|option 4|option 5’/
’E=1|2|3|4|5’/
’R=%s checked list box choice’/
’O=k2.inc’/;
putClose;
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;


is not correct: the \$call is handled at compile time, before the PUT statement has done its work. However, one could use the following:

\$echo 'T=checklistbox' > asktest.opt
\$echo 'M=Choose multiple options' >> asktest.opt
\$echo 'D=option 1|option 2|option 3|option 4|option 5' >> asktest.opt
\$echo 'E=1|2|3|4|5' >> asktest.opt
\$echo 'R=%s checked list box choice' >> asktest.opt
\$echo 'O=k2.inc' >> asktest.opt
\$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;

or better:

\$onEcho > asktest.opt
T=checklistbox
M=Choose multiple options
D=option 1|option 2|option 3|option 4|option 5
E=1|2|3|4|5
R=%s checked list box choice
O=k2.inc
$offEcho
\$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;

6.4.4 Combo Box

The next type is T=combobox which also allows a single selection:

* import a number through a combo box
\$call =ask T=combobox M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" E="1|2|3|4|5"
$include n4.inc
display n4;
As an example consider the case where the model comes with three data sets: a small one, a medium sized one and a large data set. Each data set is stored in a separate include file: `small.inc`, `medium.inc` and `large.inc`. We want to ask the user which data set should be used and the correct include file should be used. This can be accomplished by:

```
$echo 'T=combobox' > ask.opt
$echo 'M=choose data set' >> ask.opt
$echo 'D=Small data set|Medium data set|Large data set' >> ask.opt
$echo 'E=small.inc|medium.inc|large.inc' >> ask.opt
$echo 'R=$include %s' >> ask.opt
$echo 'O=dataset.inc' >> ask.opt
$call =ask @ask.opt
$include dataset.inc
```

Newer GAMS systems allow:

```
$onEcho > ask.opt
T=combobox
M=choose data set
D=Small data set|Medium data set|Large data set
E=small.inc|medium.inc|large.inc
R=$include %s
O=dataset.inc
$offEcho
$call =ask @ask.opt
$include dataset.inc
```

In this case `dataset.inc` will contain a single line: another include statement, which is either `$include small.inc`, `$include medium.inc` or `$include large.inc`.  

---

As an example consider the case where the model comes with three data sets: a small one, a medium sized one and a large data set. Each data set is stored in a separate include file: `small.inc`, `medium.inc` and `large.inc`. We want to ask the user which data set should be used and the correct include file should be used. This can be accomplished by:

```
$echo 'T=combobox' > ask.opt
$echo 'M=choose data set' >> ask.opt
$echo 'D=Small data set|Medium data set|Large data set' >> ask.opt
$echo 'E=small.inc|medium.inc|large.inc' >> ask.opt
$echo 'R=$include %s' >> ask.opt
$echo 'O=dataset.inc' >> ask.opt
$call =ask @ask.opt
$include dataset.inc
```

Newer GAMS systems allow:

```
$onEcho > ask.opt
T=combobox
M=choose data set
D=Small data set|Medium data set|Large data set
E=small.inc|medium.inc|large.inc
R=$include %s
O=dataset.inc
$offEcho
$call =ask @ask.opt
$include dataset.inc
```

In this case `dataset.inc` will contain a single line: another include statement, which is either `$include small.inc`, `$include medium.inc` or `$include large.inc`.  

---
6.4.5 List and Checklist Box

In some cases it may be useful to be able to select multiple items from a list. This can be done with `T=listbox`:

* import a set through a listbox
$call =ask T=listbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option 5" E="1|2|3|4|5" R="%s list box choice" O=k.inc
Set k /
$include k.inc
/;
display k;

Selecting multiple entries involves holding down the CTRL key. Sometimes a convenient alternative is `T=checklistbox`:

* import a set through a checked listbox
$call =ask T=checklistbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option 5" E="1|2|3|4|5" R="%s 'checked list box choice'" O=k2.inc
Set k2 /
$include k2.inc
/;
display k2;
When we select options 1, 3 and 5, the following include file is generated:

* Ask Import Filter Version 1.1, Aug. 2002
* Erwin Kalvelagen, GAMS Development Corp.
1 'checked list box choice'
3 'checked list box choice'
5 'checked list box choice'

which will populate the set k2 as follows:

\begin{verbatim}
1 * import a set through a checked listbox
3 set k2 /
INCLUDE D:\GAMS PROJECTS\ASK\K2.INC
5 * Ask Import Filter Version 1.1, Aug. 2002
6 * Erwin Kalvelagen, GAMS Development Corp.
7 1 'checked list box choice'
8 3 'checked list box choice'
9 5 'checked list box choice'
10 /
11 display k2;
12 SET k2
1, 3, 5
\end{verbatim}

The last type is the generic string type: \texttt{T=string}. This allows the user to enter any string, which is then copied to an include file as is.

\subsection*{6.4.6 File Open Dialog Box}

The type \texttt{T=fileopenbox} will display a file open dialog box from which the user can select a file.
Related options are I=InitialDirectory and F=Filter. A complete example to ask for an include file could be:

```bash
$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$include '%s'" C="Select include file"
$include fln.inc
```

This will open a file open dialog box with the starting directory being the GAMS project/working directory (this is also where GAMS looks for include files by default). Only files with mask SELECT*.INC are shown. The file FLN.INC will contain an include statement with the file the user has selected.

A related method would be:

```bash
$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$setglobal incfile '%s'" C="Select include file"
$include fln.inc
$include %incfile%
```

where we use a $setglobal to set the macro incfile to contain the user-specified file name.

To let the user choose from a set of related GDX files, one could use something like:

```bash
$call =ask T=fileopenbox I="%system.fp%" F="myproject*.gdx" o=setgdxname.inc R="$setglobal gdxfile '%s'" C="Select GDX file"
$include setgdxname.inc
$gdxIn %gdxfile%
$load i
$load j
```

In case you want to ask for a filename of a file to be written, use the type T=filesavebox. E.g.:

```bash
$call =ask T=filesavebox I="%system.fp%" o=fln.inc R="$setglobal gdxfile '%s'" C="Specify gdx file"
$include fln.inc
Set i / a, b, c /;
execute_unload '%gdxfile%', i;
```

In case you want to ask for a directory, use the type T=selectdirectory. In this case the filter option F is ignored. E.g.:

```bash
$call =ask T=selectdirectory I="%system.fp%" o=fln.inc R="$setglobal myInputdir '%s'" C="Select directory"
$include fln.inc
$log Selected directory "%myInputdir"
```

### 6.5 CHK4UPD

The tool chk4upd checks if there is a newer version of GAMS available than the one you are running. It checks for the most recent version available as well as for the newest version you can run with your license in case your maintenance and support is expired.

By default it reads gamslice.txt located in a GAMS standard location. Alternatively, you can specify another license on the command line. In addition to the command line tool you will find an integration in GAMS Studio and the GAMS IDE in the Help menu.
6.6 CHOLESKY

CHOLESKY calculates the Cholesky decomposition of a symmetric positive definite matrix. Matrix decomposition \( A=LL^T \).

6.6.1 Usage

cholesky gdxin i a gdxout L

where

- **gdxin**
  - name of gdxfile with matrix
- **i**
  - name of set used in matrix
- **a**
  - name of 2 dimensional parameter inside gdxin
- **gdxout**
  - name of gdxfile for results (factor L)
- **L**
  - name of 2 dimensional parameter inside gdxout

Calculates the Cholesky decomposition \( A=LL^T \) of a symmetric positive definite matrix \( A=a(i,j) \) where \( i \) and \( j \) are aliased sets. L will contain the Cholesky factor \( L(i,j) \).

6.6.2 Example

```
$onText
Finds the Cholesky decomposition \( A=LL' \) of a positive definite symmetric matrix
\( A \) through an external program.

Erwin Kalvelagen, may 2008
$offText

Set i / i1*i5 /;

Alias (i,j);

Table a(i,j) 'original matrix'
      i1   i2   i3   i4   i5
  i1   64   48   24   8   8
  i2   48   72   42  54  36
  i3   24   42   89 107  95
  i4   8   54  107 210 186
  i5   8   36   95 186 187;
```
Parameter L(i,j) 'cholesky factor';

execute_unload 'a.gdx', i, a;
execute '=cholesky.exe a.gdx i a b.gdx L';
execute_load 'b.gdx', L;

display a, L;

* * only lower triangular part of A is used *

Table a2(i,j) 'original matrix'
    i1 i2 i3 i4 i5
  i1  64
  i2  48  72
  i3  24  42  89
  i4   8  54 107 210
  i5   8  36  95 186 187;

Parameter L2(i,j) 'cholesky factor';

execute_unload 'a.gdx', i, a2;
execute '=cholesky.exe a.gdx i a2 b.gdx L2';
execute_load 'b.gdx', L2;

display a2, L2;

6.7 CSDP

The semidefinite programming CSDP solver from COIN-OR. The communication with CSDP requires the setup of matrix data structures in a CSDP input file. In a sense a GAMS model functions as a matrix generator.

6.7.1 Usage

csdp <input problem> [<final solution>] [<initial solution>]

<input problem>
    the name of a file containing the SDP problem in SDPA sparse format.

<final solution>
    the optional name of a file in which to save the final solution.

<initial solution>
    the optional name of a file from which to take the initial solution.

6.8 CSV2GDX

CSV2GDX is a tool that reads a CSV file and writes to a GDX file. There are multiple ways to read CSV files (Comma Separated Values) inside GAMS (see Data Exchange with Text Files for instance), but a number of features available in CSV2GDX make it possible to read a CSV file where GAMS itself cannot be used. In addition to the syntax explanation and the basic functionalities demonstrated on some examples, this tutorial also discusses some advantages and disadvantages of CSV2GDX compared to the GAMS internal table statement.

6.8.1 Usage

CSV2GDX is called by specifying the CSV file and several options to define how to read the data.

csv2gdx filename {options}

Filename

The input file; the .csv file extension is assumed when no extension has been specified.

Parameters can also be read from a text file; the use of an external file for parameters is indicated by preceding the file name with a @ (at sign). When reading parameters from a text file, lines starting with an asterisk (*) will be ignored. See also Example 8.

6.8.2 Options

The following options can be used when calling CSV2GDX:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptBadUels</td>
<td>N</td>
<td>Indicate if bad UELs are accepted or result in an error return code.</td>
</tr>
<tr>
<td>autoCol</td>
<td>none</td>
<td>Generate automatic UELs for each column.</td>
</tr>
<tr>
<td>autoRow</td>
<td>none</td>
<td>Generate automatic UELs for each row.</td>
</tr>
<tr>
<td>checkDate</td>
<td>N</td>
<td>Write GDX file only if the CSV file is more recent than the GDX file.</td>
</tr>
<tr>
<td>colCount</td>
<td>none</td>
<td>Number of columns in the input file.</td>
</tr>
<tr>
<td>decimalSep</td>
<td>Period</td>
<td>Specify a decimal separator.</td>
</tr>
<tr>
<td>fieldSep</td>
<td>Comma</td>
<td>Specify a field separator.</td>
</tr>
<tr>
<td>id</td>
<td>none</td>
<td>Identifier for the symbol in the GDX file.</td>
</tr>
<tr>
<td>index</td>
<td>none</td>
<td>Identify columns to get UELs from.</td>
</tr>
<tr>
<td>output</td>
<td>&lt;CSVFilename&gt;.gdx</td>
<td>Optional output filename.</td>
</tr>
<tr>
<td>password</td>
<td>none</td>
<td>Password for an encrypted input file.</td>
</tr>
<tr>
<td>storeZero</td>
<td>N</td>
<td>Indicate how zero values are handled.</td>
</tr>
<tr>
<td>text</td>
<td>none</td>
<td>Specify the column to get explanatory text from.</td>
</tr>
<tr>
<td>trace</td>
<td>1</td>
<td>Controls the amount of information written to the log.</td>
</tr>
<tr>
<td>useHeader</td>
<td>N</td>
<td>Indicate if the first row is a header row.</td>
</tr>
<tr>
<td>value</td>
<td>none</td>
<td>Specify the column to get the values from.</td>
</tr>
<tr>
<td>valueDim</td>
<td>N</td>
<td>Adds an extra dimension for values.</td>
</tr>
<tr>
<td>values</td>
<td>none</td>
<td>Specify the columns to get the values from.</td>
</tr>
</tbody>
</table>
Note

- The user has to specify an identifier for the symbol in the GDX file within the parameter `id`, regardless of the data structure in the CSV file.
- `CSV2GDX` determines the number of columns in the CSV file from the header row or from the user defined input specified in the parameter `colCount`. Therefore, the user must enable the option `useHeader` or has to specify the number of columns within `colCount` in any case.

Some more detailed remarks on the options:

`acceptBadUels = boolean (default=N)`

This option specifies how to proceed, when a bad UEL (e.g. too long) is encountered. If set to N, reading is stopped, and an error is signaled. When set to Y, a valid UEL is made up and reading is continued.

`autoCol = string`

Generate automatic UELs for each column. The autoCol string is used as the prefix for the column label numbers. This option overrides the use of a header row. However, if there is a header row, one must skip the row by enabling `useHeader`. This option is demonstrated in Example 3.

`autoRow = string`

Generate automatic UELs for each row. The autoRow string is used as the prefix for the row label numbers. The generated unique elements will be used in the first index position shifting other elements to the right. Using `autoRow` can be helpful when there are no labels that can be used as unique elements but also to store entries that would be a duplicate entry without an unique row label. This option is demonstrated in Example 3.

`checkDate = boolean (default=N)`

Write GDX file only if the CSV file is more recent than the GDX file to save resources when running the model containing the `CSV2GDX` call multiple times. This option is demonstrated in Example 5.

`colCount = integer`

Number of columns in the input file. This parameter is required if there is no header row, since `CSV2GDX` determines the number of columns from the header row. This option is demonstrated in Example 3. Note that the `lastCol` constant cannot be used for the `colCount` option.

`decimalSep = [Period, Comma] (default=Period)`

Specify a decimal separator. The decimal is normally a period, but this parameter allows a comma as the decimal separator, too. Special values recognized are `Eps`, `NA` resp. `N/A`, `Inf`, `True`, `False`, `None`, `Null` and `Undef` (case insensitive). A string that is not recognized as a valid number will be stored as `Undef`. This option is demonstrated in Example 2 and Example 6 (focusing on reading special values) for instance.

`fieldSep = [Comma, SemiColon, Tab] (default=Comma)`
Specify a field separator. Fields are normally separated by a comma, but this parameter allows some additional choices. Using tabs as delimiter should be avoided, since text editors act different on handling them. For instance, one must specify tabs in the GAMS IDE explicitly using %system.tab%:

```
$onEcho > tabSeparated.csv
USA%system.tab%100
GER%system.tab%70
$offEcho
```

```
$call csv2gdx tabSeparated.csv id=x fieldSep=tab index=1 colCount=2 value=lastCol
```

This option is demonstrated in Example 2 and Example 3 for instance.

**id** = *string*

Identifier for the symbol in the GDX file. Additional symbols, Dim1, Dim2, ... for the domain sets of the symbol id will be added automatically to the GDX file. Executing CSV2GDX without specifying an identifier will fail.

**index** = *list of columns*

Identify columns to get UELs from. The columns are represented as a list of integers separated by comma. For example `index=1,2,3,4` resp. `index=(1,2,3,4);` in this case the notation `1..4` is allowed. Brackets can only be used on Windows systems. The index option is used in all examples.

**output** = *filename* (default=<CSVFilename>.gdx)

Optional output filename. If no output file is specified, CSV2GDX will use the input filename and change the file extension to .gdx. If no path is specified, the output file will be created in the current directory. This option is demonstrated in Example 2 for instance.

**password** = *string*

Password for an encrypted input file. Use ENDECRYPT to encrypt a file. This option is demonstrated in Example 7.

**storeZero** = *boolean* (default=N)

Indicate if zero values are ignored or written as EPS; an empty field is always ignored. This option is demonstrated in Example 6.

**text** = *integer*

Specify the column to get explanatory text from when reading a set. For example text=5. This option is demonstrated in Example 9.

**trace** = *integer* (default=1)

Controls the amount of information written to the log. Higher values will generate more output. Valid range is 0..3. Set `trace=0` to prevent writing any information to the log. This feature is demonstrated in Example 6.

**useHeader** = *boolean* (default=N)
Indicate if the first row is a header row. The fields in the header row of the columns specified within the values option will be used as UELs. A header row is not needed or should be ignored when using the colCount or autoCol option. To skip one existing header row while using autoCol, enable useHeader. This option is demonstrated in Example 1 and Example 2 for instance.

\[ \text{value} = \text{integer} \]

Specify the column to get the values from. For example \( \text{value}=5 \). This option is demonstrated in Example 2.

\[ \text{valueDim} = \text{boolean} \text{ (default=N)} \]

Indicate if an extra dimension for values is added to the parameter even if there is just one value column. This is ignored, if there is no value column. This feature is demonstrated in Example 2.

\[ \text{values} = \text{list of columns} \]

Specify the columns to get the values from. When using a list of columns for the values and useHeader enabled, each field in the first row of the columns is used as UEL to identify the values in those columns. See also useHeader and autoCol below. If the number of columns is unknown, the symbolic constant lastCol can be useful: \( \text{values}=2..\text{lastCol} \). This option is demonstrated in Example 1 and Example 5 for instance.

### 6.8.3 Advances and limitations

**Advances**

- CSV2GDX enables the user to read CSV data where the table statement cannot be used without doing further preprocessing, e.g. in case of semicolon- or tab separated data or comma separated decimals.
- In general, CSV2GDX is a quite performant tool compared to the GAMS internal table statement.

**Limitations**

- Suppose we want to skip some rows while reading the data. For example, if the CSV file contains some reference information in a specific row which we do not want to be stored in the domain sets or in the parameter. However, skipping rows cannot be done with CSV2GDX.
- There might be CSV files with no header and varying, unknown length of rows. Since CSV2GDX determines the number of columns based on the header row or by setting the colCount option in advance, CSV2GDX might return incorrect results or the execution is aborted.
- Reading several parameters from a CSV file cannot be done directly within the CSV2GDX call. The data must be split later on as demonstrated in Example 4.
- CSV2GDX cannot read CSV files containing line breaks within a (quoted) field. You will either get an error message - "Quoted field not terminated with closing quote" - or the result might not be correct for unquoted fields, because the field content will be cut by the line break.
6.8.4 Getting Started

We introduce the basic functionalities of CSV2GDX on some simple examples. Note that many CSV files can be read inside GAMS directly using a table statement, but a number of features available in CSV2GDX enable the user to read a CSV file where the table statement cannot be used, e.g. reading files with semicolon separated data or if the decimals are separated by comma instead of a period.

6.8.4.1 Example 1 - Reading CSV Files with CSV2GDX

The first example of this collection demonstrates the key commands of CSV2GDX. For instance, consider the table statement of the model [transport] from the GAMS Model Library:

\[
\text{Table } d(i,j) \text{ 'distance in thousands of miles'}
\]

\[
\begin{array}{ccc}
\text{seattle} & 2.5 & 1.7 & 1.8 \\
\text{san-diego} & 2.5 & 1.8 & 1.4 \\
\end{array}
\]

The data can be stored in distance.csv like this:

, new-york, chicago, topeka
seattle, 2.5, 1.7, 1.8
san-diego, 2.5, 1.8, 1.4

First of all, CSV2GDX is called now, to generate distance.gdx by processing the input file distance.csv:

```
csv2gdx distance.csv id=d index=1 values=2..lastCol useHeader=y
```

CSV2GDX generates one single parameter \( d \) and two domain sets from the input file. The name of the parameter in the GDX file must be declared within the id option, while the domain sets for this parameter will be labeled with Dim1 and Dim2 automatically. Column number one is specified as the first domain set within the index option. The values option is used to specify the column numbers 2, 3, 4 containing the data values. By enabling the useHeader option, the fields of the first row of the columns specified within the values option will be handled as the second domain set. If the number of columns is unknown in advance, one can use the lastCol constant in the values or index option.

However, to complete the declaration of the sets and parameter for the model transport, one must load the data from distance.gdx:
Set
  i 'canning plants'
  j 'markets';

.gdxIn distance.gdx
.load i = Dim1
.load j = Dim2

Parameter d(i,j) 'distance in thousands of miles';
.load d
.gdxIn
display i, j, d;

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx2] for reference.

6.8.4.2 Example 2 - Reading Semicolon separated Data

In this example, the distances from the previous example are stored as a list. We want to read the cities and the column containing the miles measurement. Note, that the fields are separated by semicolon and the decimals by comma. As described in Data Exchange with Text Files the CSV file must be preprocessed with the POSIX tools in advance to replace commas by dots and semicolons by commas to read the data directly using a simple table statement.

    i;j;miles
    seattle;new-york;2,5
    seattle;chicago;1,7
    seattle;topeka;1,8
    san-diego;new-york;2,5
    san-diego;chicago;1,8
    san-diego;topeka;1,4

The data will be stored as distanceOut.gdx by adding the output file option. One can specify the field and decimal separators within the fieldSep and decimalSep option. Domain sets for the parameter to be read are declared by index=1,2.

    csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHeader=y

In order to load the data from the GDX file, execute the commands from the previous example. However, note that Dim2 does not contain the UELs from the header row this time, but the unique elements of the second column specified in the index option. The option useHeader is enabled to indicate that there is a header row to be skipped when reading the values. Also note, that the symbol d in the GDX file has exactly two dimensions. It may be useful to add an additional dimension to the parameter dmod, e.g. if different measurement units for the distances may become relevant later on and need to be calculated (kilometer for instance).

This can be done by adding the option valueDim:

    csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHeader=y valueDim=y
The option adds a third dimension to \( d \). Now we want to add the distances in kilometer by calculating the values inside the model:

\[
\begin{align*}
\text{Set } m & \text{'measurement unit' / miles, km /; } \\
\text{Parameter } \text{dmod}(i,j,m); \\
$\text{gdxIn distanceOut.gdx}$ \\
$\text{load dmod = d}$ \\
$\text{gdxIn}$ \\
\text{display dmod; } \\
\text{dmod}(i,j,'km') = 1.852*\text{dmod}(i,j,'miles'); \\
\text{display dmod; } \\
\text{Parameter } d (\text{valueDim disabled}): \\
\text{---- 36 PARAMETER } d \text{ distance in thousands of miles } \\
\text{new-york} & \text{ chicago} & \text{ topeka} \\
\text{seattle} & 2.500 & 1.700 & 1.800 \\
\text{san-diego} & 2.500 & 1.800 & 1.400 \\
\text{Parameter } \text{dmod resp. } d \text{ with valueDim enabled before calculating further measurements: } \\
\text{---- 49 PARAMETER } \text{dmod} \\
\text{miles} \\
\text{seattle .new-york} & 2.500 \\
\text{seattle .chicago} & 1.700 \\
\text{seattle .topeka} & 1.800 \\
\text{san-diego.new-york} & 2.500 \\
\text{san-diego.chicago} & 1.800 \\
\text{san-diego.topeka} & 1.400 \\
\text{Parameter } \text{dmod with valueDim enabled after calculating the distances in kilometer inside the GAMS model: } \\
\text{---- 52 PARAMETER } \text{dmod} \\
\text{miles km} \\
\text{seattle .new-york} & 2.500 & 4.630 \\
\text{seattle .chicago} & 1.700 & 3.148 \\
\text{seattle .topeka} & 1.800 & 3.334 \\
\text{san-diego.new-york} & 2.500 & 4.630 \\
\text{san-diego.chicago} & 1.800 & 3.334 \\
\text{san-diego.topeka} & 1.400 & 2.593 \\
\end{align*}
\]

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx3] for reference.
6.8.4.3 Example 3 - Dealing with missing Labels and Duplicates

Missing Labels

The file EUCData.csv contains the extracted euclidean coordinates of the first nine cities of berlin52.tsp from TSPLib. You might want to import the data to calculate a complete distance matrix inside GAMS to find an optimal traveling salesman tour for instance.

<table>
<thead>
<tr>
<th>City</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>565.0;575.0</td>
</tr>
<tr>
<td></td>
<td>25.0;185.0</td>
</tr>
<tr>
<td></td>
<td>345.0;750.0</td>
</tr>
<tr>
<td></td>
<td>945.0;685.0</td>
</tr>
<tr>
<td></td>
<td>845.0;655.0</td>
</tr>
<tr>
<td></td>
<td>880.0;660.0</td>
</tr>
<tr>
<td></td>
<td>25.0;230.0</td>
</tr>
<tr>
<td></td>
<td>525.0;1000.0</td>
</tr>
<tr>
<td></td>
<td>580.0;1175.0</td>
</tr>
</tbody>
</table>

There is no header row, neither a column with labels to serve as domain sets for the coordinates of the cities. However, CSV2GDX automatically generates UELs for columns and rows by adding ascending numbers to an user-defined prefix specified within the autoCol and autoRow option. Also note, because of the missing header, the number of columns in the file must be manually determined and declared within `colCount`.

```gams
    csv2gdx EUCData.csv id=coord fieldSep=semiColon autoCol=x autoRow=city colCount=2 values=1,2
```

The rows will be labeled with `city1...city9`, the columns with `x1` and `x2`.

Load the parameter and sets from the GDX file to calculate the complete distance matrix. The set `Dim1` contains the UELs `city1...city9` for the set `i` of cities, `Dim2` is the set of axes/coordinates i.e. containing the elements `x1` and `x2` to represent the x-axis and y-axis:

```gams
    Set
      i   'cities'
      axes 'x1 and x2 axes';
    $gdxIn EUCData.gdx
    $load i   = Dim1
    $load axes = Dim2
    Parameter coord(i,axes) 'coordinate of city i';
    $load coord
    $gdxIn
    display coord;
```

The display statement generates the following output:

```plaintext
    ---- 43 PARAMETER coord  coordinate of city i
           x1        x2
          city1 565.000  575.000
          city2  25.000  185.000
```
One could now easily proceed calculating a complete distance matrix:

Alias (i,j);

Parameter c(i,j) 'euclidean distance between city i and j';
c(i,j) = eDist(coord(i,"x1") - coord(j,"x1"),coord(i,"x2") - coord(j,"x2"));

Duplicates

In the previous example, using the autoCol and autoRow option had an additional benefit as we declared the set of the cities on the fly. However, one major advantage/purpose of these options is to prevent error messages or loss of data when reading rows with duplicate keys.

Consider the input file duplicates.csv:

red,red,1
red,red,2
red,green,3
blue,blue,4

Note the duplicate key in the first two rows. By the use of the autoRow parameter in the CSV2GDX call unique labels are added to each row. This way a GAMS program can store all data with duplicate keys and prepare for better error messages.

csv2gdx duplicates.csv id=data index=1,2 value=3 colCount=3 autoRow=row

Ascending numbers will be added to the 'row' prefix specified.

The data can be easily loaded into sets and parameter by executing the following lines:

Set
   row 'UELs generated by autoRow'
   color 'set of colors';

$gdxIn duplicates.gdx
$load row = Dim1
$load color = Dim2
$loadm color = Dim3

Parameter data(row,color,color);
$load data
$gdxIn
display row, color, data;
Note the usage of the `$loadm` command to merge all colors into one set of colors. The display statement generates the following output in the listing file:

```plaintext
---- 77 SET row UELs generated by autoRow
row1, row2, row3, row4

---- 77 SET color set of colors
red , blue , green

---- 77 PARAMETER data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>blue</td>
<td>green</td>
</tr>
<tr>
<td>row1.red</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>row2.red</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>row3.red</td>
<td>3.000</td>
<td></td>
</tr>
<tr>
<td>row4.blue</td>
<td>4.000</td>
<td></td>
</tr>
</tbody>
</table>
```

The complete example is also part of the GAMS Data Utilities Library, see model [csv2gdx4] for reference.

6.8.4.4 Example 4 - Reading several Parameters from a single Input File

This example demonstrates how to read the data from a single input file into different parameters. However, this cannot be done directly using the CSV2GDX tool, since CSV2GDX writes to a single parameter (see section limitations). For instance, an energy supplier plans to build new transmission lines from their power plants to some distribution stations. There are different stages for the transmissions lines. The relevant data to be read are the plant and station identifiers, the capacity bounds per transmissions line on a certain stage and the associated cost. The data is all stored in a single file:

```
plant;station;length;minCap;maxCap;stage;cost
p1;s1;100;50;100;1;1200
p1;s2;75;35;65;1;1400
p1;s1;100;100;150;2;1800
p2;s1;150;50;100;1;1400
p2;s1;150;100;150;2;2000
p2;s1;150;150;200;3;2350
p2;s2;75;25;50;1;600
p2;s2;75;50;75;2;800
p3;s1;80;40;100;1;1050
```

Read networkData.csv with CSV2GDX by calling:

```
csv2gdx networkData.csv id=dataPar useHeader=y fieldSep=semiColon index=1,2,6 values=4,5,7
```
Note that the stage stored in column six is a domain set, too, while the length of the transmission line in the third column is of no interest. Since CSV2GDX writes to a single parameter, one must split the data later on into minCap, maxCap and cost for instance:

Parameter
dataPar
  minCap(plant,station,stage)
  maxCap(plant,station,stage)
  cost(plant,station,stage);

$gdxIn networkData.gdx
$load plant = Dim1
$load station = Dim2
$load stage = Dim3
$load dataPar
$gdxIn

minCap(plant,station,stage) = dataPar(plant,station,stage,'minCap');
maxCap(plant,station,stage) = dataPar(plant,station,stage,'maxCap');
cost(plant,station,stage) = dataPar(plant,station,stage,'cost');

This example is also part of the GAMS Data Utilities Library, see model \[\text{csv2gdx5}\] for reference.

6.8.4.5 Example 5 - Reading economic Data from the World Bank Data Catalog

This example demonstrates how to read some real CSV data from the World Bank Data Catalog. Suppose we want to read some time series data, e.g. GDP growth rates. The data is structured as follows (rows shortened for presentation):

"East Asia and Pacific","EAA","GDP growth, constant 2010 USD","NYGDPMKTPKDZ","","","","","","","","","","","","","","","","","","","","","",

We are not interested in the "Country Code", "Indicator Name" and "Indicator Code". Since the annual GDP rate per country is unique, the CSV2GDX call is quite easy despite the large number of columns. Note that there are only a few limitations of CSV2GDX, discussed in the section Advances and limitations. Empty fields in the data will always be ignored, causing no trouble at all if the field separators are set correctly.

csv2gdx GDPData.csv id=GDPG index=1 values=5..lastCol useHeader=y checkDate=y

The option checkDate is enabled to save resources if you run the model multiple times, as the GDX file is only written if the CSV file is more recent than the GDX file. To load the sets and parameter from the GDX file, execute the following commands:

Set country, year;
$gdxIn GDPData.gdx
$load country = Dim1
$load year = Dim2

Parameter GDPRate(country,year);
$load GDPRate
$gdxIn
display country, year, GDPRate;
The display statement generates the following output in the listing file. Obviously, double quotes are removed from the fields of the value columns, but also from the index columns:

```
---- 12 SET country

East Asia and Pacific, Europe and Central Asia, Latin America and the Caribbean, Middle East and North Africa, South Asia, Sub-Saharan Africa, World (WBG members)
Afghanistan, Albania, Algeria, Angola

---- 12 SET year


---- 12 PARAMETER GDPR

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe and Central Asia</td>
<td>0.963</td>
<td>1.672</td>
<td>3.782</td>
<td>2.901</td>
<td>3.029</td>
<td>2.969</td>
</tr>
<tr>
<td>Latin America and the Caribbean</td>
<td>-0.562</td>
<td>-1.526</td>
<td>0.902</td>
<td>2.044</td>
<td>2.586</td>
<td>2.701</td>
</tr>
</tbody>
</table>
```

Note that the listings have been shortened for presentation. However, there is no data for the years 1999-2014.

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx6] for reference.

### 6.8.5 Additional Examples for extended Use

The examples in this section discuss some special features. Some topics were already mentioned briefly in the previous section like reading compressed and encrypted files or reading the CSV2GDX options from an external file.

#### 6.8.5.1 Example 6 - Reading special Values

To illustrate how special values are interpreted, consider the following data:

```
one,two,three,four,five,six
red,red,,Undef,'3.3',red
red,red,"4.4",5.5,Eps,green
"red","green",7.7e+02,8.8,-Inf,blue
blue,blue,10,0,NA,purple
brown,blue,true,false,N/A,green
black,red,None,Null,"True",blue
```

Calling CSV2GDX to read the data and write to GDX:

```
csv2gdx data.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3
```
The GAMS log reports three occurrences of undefined values. Note that the amount of information written to the log was increased by setting the option `trace=3`.

---
call csv2gdx specialValues.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3

CSV2GDX 25.2.0 r67638 ALFA Released 15Aug18 WEI x86 64bit/MS Windows

Header enabled, number of columns = 6
1: one
2: two
3: three
4: four
5: five
6: six
2: |red|, |red|, | |, |Undef|, |'3.3'|, |red|
3: |red|, |red|, |4.4|, |5.5|, |Eps|, |green|
4: |red|, |'green'|, |7.7e+02|, |8.8°|, |-Inf|, |blue|
5: |blue|, |blue|, |10|, |0|, |NA|, |purple|

Undef Count = 3, No errors, CSV2GDX time = 157ms

As mentioned in decimalSep, CSV2GDX fails to recognize the number 8.8 from the string 8.8° because of the unknown special character, while 7.7e+02 is interpreted as a number of course. Watch out if there are quotes in the values or index columns. The number 4.4 enclosed by double quotes is interpreted as a number, while '3.3' is not (as you can see in the log, the double quotes are eliminated, while the single quotes remain). In the first column, the double quotes enclosing the string red are removed, while the single quotes in the second column enclosing the string green are not removed. The zero value is stored as EPS by setting `storeZero=y`. Note that no error was reported, even though some of the values stored as `Undef` may cause some trouble in your model later on (use `$onUndf` to enable loading parameters with undefined values).

The booleans `true` and `false` in the second last row are represented by the numericals 1 resp. 0 within GAMS. The special values `None` and `Null` will be converted to 0. Since `storeZore` is enabled in this example, the value displayed for `false`, `None` and `Null` is EPS.

Suppose we want to declare the parameter A as: A(color,color,color,number). We can proceed as follows:

```plaintext
Set color, number;
$gdxIn data.gdx
$load color = Dim1
$loadm color = Dim2
$loadm color = Dim3
$load number = Dim4

Parameter A(color,color,color,number);
$onUndf
$load A
$offUndf
$gdxIn
display A;
```

Loading undefined values is enabled by adding `$onUndf`. This example is also part of the GAMS Data Utilities Library, see model [csv2gdx7] for reference.
6.8.5.2 Example 7 - Reading a compressed encrypted Input File

Reading a compressed input file is supported by CSV2GDX. The gzip program in the gbin sub-directory or ENDECRYPT must be used for compression. Call gzip to compress the CSV file by running the following command:

```
gzip compressMe.csv -c > compressedFile.csv.gz
```

Gzip writes to standard output by appending -c to keep the original file unchanged. The output is then redirected to compressedFile.csv.gz. To read the compressed file, call CSV2GDX with the same options as for processing the uncompressed file.

```
csv2gdx compressedFile.csv.gz output=unCompressedGzipFile.gdx id=d index=1 useHeader=y values=2..last
```

The data in this example is taken from Example 1.

While the main purpose of Endecrypt is about encrypting and decrypting files, it also compresses the file. If you want to compress a file with Endecrypt, do not specify an password file:

```
cat compressMe.csv | endecrypt -W compressedFile.csv
```

The option -W encrypts standard input and writes to compressedFile.csv. Although there will be no encryption because of the missing password file, one must set the option -W. You can read the compressed file with CSV2GDX:

```
csv2gdx compressedFile.csv output=unCompressedEndycryptFile.gdx id=d index=1 useHeader=y values=2..last
```

Note that there is no further file extension added.

By adding a password file (containing the password in the first line), Endecrypt encrypts and compresses the input file:

```
cat compressMe.csv | endecrypt -W compressedEncryptedFile.csv passwordFile.txt
```

The password file will be deleted. Execute the following command to read the compressed and encrypted file with CSV2GDX:

```
csv2gdx compressedEncryptedFile.csv output=unCompressedDecryptedFile.gdx password=Anton id=d index=1
```

The password option is added. You must set the password directly within the option, in this case “Anton”, not the password file. Note that CSV2GDX does not support .gz, .7z etc. compressed files!

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx8] for reference.
6.8.5.3 Example 8 - Reading Options from an external File

This example demonstrates how to read the options from an external text file as already mentioned in section Filename. The file distance.csv from the Getting Started Example 1 will be processed with CSV2GDX while reading the options from an external text file named howToRead.txt.

\[ \text{csv2gdx distance.csv @howToRead.txt} \]

Swapping the order of the CSV filename and the instructions filename inside the call statement will cause an error. Note the leading @ (at sign) on the instructions file, containing the following options:

* These lines are interpreted as a comment
* This file specifies the options for reading distance.csv using CSV2GDX

\[
\begin{align*}
\text{id} & = \text{d} \\
\text{fieldSep} & = \text{semiColon} \\
\text{decimalSep} & = \text{comma} \\
\text{index} & = \text{i} \\
\text{useHeader} & = \text{y} \\
\text{values} & = 2..\text{lastCol}
\end{align*}
\]

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx9] for reference.

6.8.5.4 Example 9 - Reading Set Elements with explanatory Text

In this example, we will demonstrate how to read explanatory text of set elements using the CSV2GDX option text.

Consider the input file data.csv:

\[
\begin{align*}
\text{a1,b1,explanatory text of a1.b1,10} \\
\text{a1,b2,explanatory text of a1.b2,20} \\
\text{a2,b1,explanatory text of a2.b1,30} \\
\text{a2,b2,explanatory text of a2.b2,40}
\end{align*}
\]

The set elements are stored in the first and second column, the explanatory text is stored in the third column and there is a fourth column containing some values. Suppose we want to read a two-dimensional set with explanatory text. By default, CSV2GDX does not read explanatory text, i.e. by running the following command:

\[ \text{csv2gdx data.csv id=abOnlyUELs index=1,2 colCount=4} \]

CSV2GDX creates a GDX file containing the two-dimensional set abOnlyUELs without explanatory text (needless to say, we did not specify a column to get the explanatory text from). By specifying the third column within the index option, i.e. index=1..3, the result will be a three-dimensional set (without explanatory text, too). Using the value option is not suitable, as the data type will be a parameter instead of a set (in addition, CSV2GDX expects numeric data, potentially leading to undefined values). We can read the explanatory text easily by specifying column three within the text option:

\[ \text{csv2gdx data.csv id=abWithExpText index=1,2 text=3 colCount=4} \]

The figure shows the set abOnlyUELs on the left without explanatory text and the set abWithExpText on the right:

Note that the text and value(s) options cannot be used at the same time (instead, use multiple CSV2GDX calls in scenarios when you wish to read set elements with explanatory text and parameters from a single data set).
6.9 EIGENVALUE

EIGENVALUE calculates eigenvalues of a symmetric matrix.

6.9.1 Usage

eigenvalue gdxin i a gdxout ev

where

  gdxin
    name of gdxfile with matrix

  i
    name of set used in matrix

  a
    name of 2 dimensional parameter inside gdxin

  gdxout
    name of gdxfile for results (eigenvalues)

  ev
    name of 1 dimensional parameter inside gdxout

Calculates eigenvalues of symmetric matrix \( a(i,j) \) where \( i \) and \( j \) are aliased sets.

6.9.2 Example

$onText
Eigenvalue example.

octave:1> a=[9 1 1; 1 9 1; 1 1 9]
a =
   9  1  1
   1  9  1
   1  1  9

octave:2> eig(a)
an =
   8
   8
  11
$offText

Set i / i1*i3 /;


Alias (i,j);

Table a(i,j)
  i1  i2  i3
  i1  9  1  1
  i2  1  9  1
  i3  1  1  9;

Parameter e(i) 'eigenvalues';

execute_unload 'mat.gdx', i, a;
execute '=eigenvalue.exe mat.gdx i a ev.gdx e';
execute_load 'ev.gdx', e;
display a, e;

6.10 EIGENVECTOR

EIGENVECTOR calculates eigenvalues/vectors of a symmetric matrix.

6.10.1 Usage

eigenvector gdxin i a gdxout eval evec

where

gdxin
  name of gdxfile with matrix

i
  name of set used in matrix

a
  name of 2 dimensional parameter inside gdxin

gdxout
  name of gdxfile for results (eigenvalues)

eval
  name of 1 dimensional parameter inside gdxout

evec
  name of 2 dimensional parameter inside gdxout

Calculates eigenvalues/vectors of symmetric matrix a(i,j) where i and j are aliased sets. eval will contain the eigenvalues and evec will contain the eigenvectors.
6.10.2 Example

Eigenvector example.

octave:1> a = [1 2 4 7 11; 2 3 5 8 12; 4 5 6 9 13; 7 8 9 10 14; 11 12 13 14 15]
a =
    1 2 4 7 11
    2 3 5 8 12
    4 5 6 9 13
    7 8 9 10 14
    11 12 13 14 15
octave:2> eig(a)
ans =
    -8.464425
    -1.116317
    -0.512109
    -0.027481
    45.120332
octave:3> [e1,e2] = eig(a)
e1 =
    0.5550905 -0.2642556 0.2892854 0.6748602 0.2879604
    0.4820641 -0.2581518 0.2196341 -0.7349311 0.3355726
    0.2865066 0.2159261 -0.8437897 0.0411896 0.3970041
    -0.0992784 0.7711236 0.3943678 0.0055409 0.4898525
    -0.6062562 -0.4714561 -0.0238286 0.0520829 0.6378888
e2 =
    -8.46442 0.00000 0.00000 0.00000 0.00000
    0.00000 -1.11632 0.00000 0.00000 0.00000
    0.00000 0.00000 -0.51211 0.00000 0.00000
    0.00000 0.00000 0.00000 -0.02748 0.00000
    0.00000 0.00000 0.00000 0.00000 45.12033

Set i / i1*i5 /;
Alias (i,j);
Table a(i,j)
    i1  i2  i3  i4  i5
    i1  1  2  4  7 11
    i2  2  3  5  8 12
    i3  4  5  6  9 13
    i4  7  8  9 10 14
    i5 11 12 13 14 15;
Parameter
eval(i) 'eigenvalues'
evec(i,j) 'eigenvectors';
executeUnload 'mat.gdx', i, a;
execute '=eigenvector.exe mat.gdx i a ev.gdx eval evec';
execute_load 'ev.gdx', eval, evec;
display a, eval, evec;

6.11 ENDECRYPT

A tool to encrypt and decrypt text files.

6.11.1 Usage

decrypt option xfile [passwdfile]

option

-W  
  to encrypt standard input and write to xfile

-R  
  to decrypt xfile and write to standard output

xfile
  Text file to encrypt or decrypt.

passwdfile (optional)

  First line is used as password; will be deleted after use

6.12 FINDTHISGAMS

6.12.1 Introduction

The Windows command line tool findthisgams is used for modifying GAMS specific registry entries created by the GAMS installer. These registry entries are primarily used for Windows file associations for GAMS related files (*.gms, *.gdx, *.gpr) and for the Object-oriented GAMS APIs which use the registry key gams.location for finding a valid GAMS system directory. This documentation uses the following registry hive abbreviations:

- HKCU: HKEY_CURRENT_USER
- HKLM: HKEY_LOCAL_MACHINE
- HKCR: HKEY_CLASSES_ROOT

Attention

This tool should be used with caution and only when necessary, as it changes the state of the registry. Reading this documentation carefully before use is advised.
6.12.2 Usage

`findthisgams.exe [-q] [ list | delete (HKLM|HKCU) | write (HKLM|HKCU) ] [ide]`

**Without arguments:** When `findthisgams.exe` is run without arguments, the tool tries to register the current GAMS system directory by either writing to HKCU (which takes precedence over HKLM) or by deleting entries in HKCU that hide the correct entries (current system directory) in HKLM.

- `q`: Run quietly. No pop-ups.
- `list`: Print a summary of GAMS related registry entries (old and new).
- `delete (HKLM|HKCU)`: Deletes the registry entries in either HKLM or HKCU. The tool tries to delete both the old and the new style registry entries.
- `write (HKLM|HKCU)`: Writes the registry entries for the current GAMS system directory either to HKLM or HKCU.
- `ide`: Per default, when `findthisgams.exe` writes to the registry, it uses GAMS Studio for file associations. Specifying `ide` will register the GAMS IDE instead.

6.12.3 Registry Keys

The registry entries created by the Windows installer have changed with GAMS 32. When running `findthisgams` with command line parameter `list`, the registry is queried for both the old and the new style registry keys. This gives a good overview of the current state of the registry before initiating any actual modifications to the registry:

**Old registry keys (up to GAMS 31):**

- `\gms`
- `\gdx`
- `\gamside.file`
- `\gamside.file\shell\open\command`
- `\gpr`
- `\gamside.project`
- `\gamside.project\shell\open\command`
- `\gams.location`

**New registry keys (GAMS 32 and later):**

- `\Software\Classes\\gms`
- `\Software\Classes\\gdx`
- `\Software\Classes\gamside.file`
- `\Software\Classes\gamside.file\DefaultIcon`
- `\Software\Classes\gamside.file\shell\open\command`
- `\Software\Classes\\gpr`
- `\Software\Classes\gamside.project`
- `\Software\Classes\gamside.project\DefaultIcon`
- `\Software\Classes\gamside.project\shell\open\command`
- `\Software\Classes\\gams.location`

**Additional registry keys (GAMS 37 and later):**

- `\gsp`
- `\Software\Classes\gamsstudio.project`
- `\Software\Classes\gamsstudio.project\DefaultIcon`
- `\Software\Classes\gamsstudio.project\shell\open\command`

**Note**

In order to access registry entries in HKLM, `findthisgams` needs to be run with administrative privileges.
6.13 GAMS Studio

GAMS Studio is a completely new integrated development environment for GAMS, which is available for Windows, Mac OS X, and Linux. It is based on C++ and Qt.

Note
In addition to this technical documentation, there is also a tutorial about the usage of GAMS Studio.

6.13.1 Motivation

The classic GAMS IDE has been shipped with the GAMS system for the last 20 years and is still the workhorse for many GAMS programmers. However, the existing IDE does not provide all the features we see in modern development environments. Due to its underlying software technology, implementing new features in the current IDE turned out to be a poor option, so work on a new development environment started from scratch. GAMS Studio is based on C++ and Qt, which makes it fast, reliable, and platform independent (Windows, Mac OS X, and Linux). An overview about the differences between GAMS Studio and the classic GAMS IDE can be seen at the end of this chapter (for long-time GAMSIDE users, especially the difference between IDE projects and Studio projects might be interesting).

6.13.2 Central Widgets

6.13.2.1 Welcome Page

The Welcome page is the starting point of GAMS Studio. It is designed to give quick access to common actions and to offer helpful information. It is divided into three columns.

On the very left there is a column labeled "Last Files" which lists all files that have been opened recently in Studio with the most recent file on top. A simple left click on an item in this list opens the corresponding file. The middle column named "Getting Started" offers useful actions and links for new users. The upper half contains shortcuts to create new files in user defined locations, open the GAMS Model Library Explorer or load the Transport example. The lower half has a link to the Studio introduction video on YouTube and two further links that open the integrated help view showing either this Studio documentation or the page with the GAMS tutorial overview. The rightmost column "Further Help" contains a link to the recent changes of GAMS Studio, the latest GAMS release notes, the GAMS World Forum for support and information about how to contact GAMS.
6.13.2.2  Code Editor

The Code Editor provides common functionality for editing with GAMS specific syntax highlighting. On the left side there is a special area displaying the line numbers and icons for links and errors generated by the compiler.

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Ctrl - C</td>
<td>Command - C</td>
<td>Copy selected text</td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl - X</td>
<td>Command - X</td>
<td>Cut selected text</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl - V</td>
<td>Command - V</td>
<td>Paste text from clipboard</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl - A</td>
<td>Command - A</td>
<td>Select whole document</td>
</tr>
<tr>
<td>Switch Comment</td>
<td>Ctrl - *</td>
<td>Command - *</td>
<td>(Un)Comment all lines in selection</td>
</tr>
<tr>
<td>Undo</td>
<td>Ctrl - Z</td>
<td>Command - Z</td>
<td>Reverts the latest text change</td>
</tr>
<tr>
<td>Redo</td>
<td>Ctrl - Y or</td>
<td>Command - Shift - Z</td>
<td>Restores a previous Undo</td>
</tr>
<tr>
<td></td>
<td>Ctrl - Shift - Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.13.2.2.1  1. Basic Text Manipulation

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove Line</td>
<td>Shift - Del</td>
<td>Command - Del</td>
<td>Removes the current line</td>
</tr>
<tr>
<td>Duplicate Line</td>
<td>Ctrl - D</td>
<td>Command - D</td>
<td>Duplicates the current line(s)</td>
</tr>
<tr>
<td>Lowercase</td>
<td>Alt - Shift - L</td>
<td>Option - Shift - L</td>
<td>Toggle selection to lower case</td>
</tr>
<tr>
<td>Action</td>
<td>Shortcut</td>
<td>macOS</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Uppercase</td>
<td>Alt - Shift - U</td>
<td>Option - Shift - U</td>
<td>Toggle selection to upper case</td>
</tr>
<tr>
<td>Tab</td>
<td>Tab</td>
<td>Tab</td>
<td>Add spaces till next tab size</td>
</tr>
<tr>
<td>Untab</td>
<td>Shift - Tab</td>
<td>Shift - Tab</td>
<td>Remove spaces till previous tab size</td>
</tr>
<tr>
<td>Indent</td>
<td>Ctrl - I</td>
<td>Command - I</td>
<td>Indent complete line</td>
</tr>
<tr>
<td>Outdent</td>
<td>Ctrl - Shift - I</td>
<td>Command - Shift - I</td>
<td>Outdent complete line</td>
</tr>
<tr>
<td>Find</td>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Open Search &amp; Replace window, filling search field with selected text if there is any.</td>
</tr>
<tr>
<td>Find Next</td>
<td>F3</td>
<td>F3</td>
<td>Jump to next search result if</td>
</tr>
<tr>
<td>Find Previous</td>
<td>Shift - F3</td>
<td>Shift - F3</td>
<td>Jump to previous search result</td>
</tr>
</tbody>
</table>

### 6.13.2.2.2 2. Advanced Text Manipulation

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match Parentheses</td>
<td>Ctrl - B</td>
<td>Command - B</td>
<td>Jump to matching parenthesis</td>
</tr>
<tr>
<td>Select Parentheses</td>
<td>Ctrl - Shift - B</td>
<td>Command - Shift - B</td>
<td>Select to matching parenthesis</td>
</tr>
<tr>
<td>Toggle Bookmark</td>
<td>Ctrl - M</td>
<td>Ctrl - M</td>
<td>Toggle bookmark</td>
</tr>
<tr>
<td>Previous Bookmark</td>
<td>Ctrl - ,</td>
<td>Ctrl - ,</td>
<td>Go to previous bookmark</td>
</tr>
<tr>
<td>Next Bookmark</td>
<td>Ctrl - .</td>
<td>Ctrl - .</td>
<td>Go to next bookmark</td>
</tr>
<tr>
<td>Toggle Folding</td>
<td>Alt - L</td>
<td>Alt - L</td>
<td>Toggles the folding state of the current section</td>
</tr>
<tr>
<td>Fold all Sections</td>
<td>Alt - O</td>
<td>Alt - O</td>
<td>Folds all foldable sections</td>
</tr>
<tr>
<td>Fold DCO Sections</td>
<td>Alt - I</td>
<td>Alt - I</td>
<td>Folds all foldable DCO sections</td>
</tr>
<tr>
<td>Unfold all Sections</td>
<td>Alt - Shift - O</td>
<td>Alt - Shift - O</td>
<td>Unfolds all foldable sections</td>
</tr>
<tr>
<td>Open Code Completer</td>
<td>Ctrl - Space</td>
<td>Ctrl - Space</td>
<td>Opens the Code Completer</td>
</tr>
<tr>
<td>Jump to include</td>
<td>Ctrl - Left Mouse Button or F2</td>
<td>Command - Left Mouse Button or F2</td>
<td>Jumps to file referenced by $include</td>
</tr>
<tr>
<td>Scroll Up</td>
<td>Ctrl - Up</td>
<td>Command - Up</td>
<td>Scroll up (keeping cursor position)</td>
</tr>
<tr>
<td>Scroll Down</td>
<td>Ctrl - Down</td>
<td>Command - Down</td>
<td>Scroll down (keeping cursor position)</td>
</tr>
<tr>
<td>Scroll Page Up</td>
<td>Ctrl - PageUp</td>
<td>Fn - Up</td>
<td>Scroll page up (keeping cursor position)</td>
</tr>
<tr>
<td>Scroll Page Down</td>
<td>Ctrl - PageDown</td>
<td>Fn - Down</td>
<td>Scroll page down (keeping cursor position)</td>
</tr>
<tr>
<td>Start of Line</td>
<td>Home</td>
<td>Command - Left</td>
<td>Go to the start of line (or lines content)</td>
</tr>
<tr>
<td>End of Line</td>
<td>End</td>
<td>Command - Right</td>
<td>Go to the end of the line</td>
</tr>
<tr>
<td>Start of Document</td>
<td>Ctrl - Home</td>
<td>Fn - Left</td>
<td>Go to the start of the document</td>
</tr>
</tbody>
</table>
### 6.13.2.2.3 Navigation & Selection

The Code Editor looks for parenthesis to the right then to the left of the cursor and marks valid and invalid nesting. Hitting Ctrl - B moves the cursor to the matching parenthesis keeping the state inside/outside. To select the block to the matching parenthesis hit Ctrl - Shift - B.

To support navigating to files included by the gams source, place the mouse cursor over the file name and press the Ctrl - Left Mousebutton or place the cursor at the filename and hit F2. If found, Studio will add the file to the current project and open it. Only plain text filenames can be detected this way, symbols are not resolved.

### 6.13.2.2.4 Code Folding

The Code Editor detects several kinds of sections in the source code that can be folded. These sections include all kinds of matching parentheses that extend over more than one line plus several $-control options that needs to come in pairs. To toggle the folding state of a section either the fold-marker can be clicked or the shortcut Alt - L can be pressed when the cursor is in a foldable line. To fold or unfold all foldable sections the main menu contains entries in the View menu. Additionally the shortcuts Alt - 0 to fold and Alt - Shift - 0 to unfold can be used.

To define individual blocks of folding the Dollar Control Options $onFold and $offFold can be added to the source.

List of $-control options that create a fold section:

- $onText / $offText
6.13 GAMS Studio

- `$OnEcho[V/S] / $OffEcho`
- `$OnPut[V/S] / $OffPut`
- `$OnExternalInput / $OffExternalInput`
- `$OnExternalOutput / $OffExternalOutput`
- `$ifThen[I/E] / $endIf`
- `$onFold / $offFold`

6.13.2.2.5 Syntax Highlighting  The GAMS Studio syntax highlighter recognizes different kinds of source code like

- `$control option` with some special treatments (like `$OffText`, `$OnText`, etc.)
- `comments`, e.g. line comments and comment block
- `keywords` like declarations or loops
- `description` as descriptive text for identifiers
- `data statements` for identifiers or table data
- ...

To get the best highlight result it is recommended to use strict GAMS syntax.

6.13.2.2.6 Code Completion  GAMS Studio offers a static code completion for fixed internal keywords known by the compiler. The code completer pops up after pressing `Ctrl - Space` and shows keywords that matches the current syntax and starts with the already typed characters of the word to the left of the cursor. Moving the cursor to the left shortens the matching pattern while moving the cursor to the right extends it. The keys `up/down`, `pageUp/pageDown`, and `home/end` are processed by the results list during the completer is visible. The selected word can be inserted with the keys `tab` or `enter/return` or by double clicking a word in the list. To hide the completer, hit `esc` or click outside of the completer window. Additionally the completer hides if there is no match for the current start of word.

6.13.2.2.7 Automatic closing of brackets and quote characters  This feature is meant to improve convenience for users typing source code. In GAMS, as in pretty much every other programming language, opening brackets or quote characters are followed at some point by a closing equivalent. So when a user types one of the following characters the matching closing character will also be inserted after the text cursor. This allows the user to easily type the content in between. When typing the closing character which should be right after the cursor the already existing character will be jumped over. So when a user types the closing character by force of habit the autoclose feature does not interfere or causes syntax errors. Also possible is to make a text selection and then type a bracket or quote character. The selected text will then be surrounded by the chosen character pair. In the GAMS Studio under `Settings > Editor > "Auto close opening brackets and quotes"` a switch is available to turn this auto-close feature off.

List of characters that trigger insertion of a closing character:

- `
  - (  
  - {  
  - [ `
Other editors also close the < character but in the GAMS language context it is used in a different way, making it unnecessary to be closed automatically (e.g. as a comparison operator). In some cases user might not like the closing character to be inserted automatically however, this is hard to predict. Other editors - like Qt Creator - check what the next character after the cursor is. Only a limited set of characters allows the automatic insertion of the closing character.

Following characters allow auto closing when being the next char after the text cursor:

- Whitespaces
- ,
- ;
- )
- ]
- {
- }

All others will prevent the insertion of characters.

6.13.2.2.8 File Encoding When opening a file Studio tries to assume the valid file-encoding. As it is not possible to safely determine the encoding the Edit > Encoding menu assists. To reload a file using another codec the reload with ... submenu can be used. With the convert to ... submenu the current file can be converted to another codec. A selection of encodings is preselected. This selection can be changed using the Select encoding submenu. It opens a dialog where the active encodings can be selected. One of them is marked as the Default encoding. Initially is is set to utf-8.

6.13.2.2.9 Bookmarks The Code Editor provides the ability to set temporary bookmarks in the text files opened in main tabs. If Studio is closed they are dropped. Also if a file tab is closed the bookmarks for this file are dropped. A bookmark is placed at the current position of the current file using the toggle bookmark key. A line in the editor can only carry one bookmark. If this line has a bookmark already the existing one is removed. The keys for next/previous bookmark allow to navigate through the bookmarks beyond file bounds.

6.13.2.3 Listing Viewer

The Listing Viewer is used for displaying a GAMS output or listing file (*.lst) which is generated when running a GAMS (*.gms) file. The Listing Viewer consists of two sides. On the left side, the content of the listing file is presented in a tree structure that allows for convenient navigation within the listing file. The right side shows the listing file itself.

By clicking on a specific item in the tree on the left hand side, the listing file will automatically jump to the corresponding location. Changes of the cursor position in the listing file are dynamically reflected in the tree by selecting the item that belongs to the current section of the listing file.
When the GAMS process returns with a compilation error the **Listing Viewer** contains links to the source file. If the source file doesn't exist anymore the link is still shown but striked out.

6.13.2.4 GDX Viewer

The GDX Viewer is used to open and inspect GDX (GAMS Data eXchange) files. A GDX file contains GAMS symbols such as sets, parameters, variables, and equations in a binary format that serves as input and output to GAMS. The GDX Viewer component consists of two sides. The left side displays the symbol table of the open GDX file in a table format. Every entry represents a symbol and consists of the following information:

- **Entry**: The index of the symbol
- **Name**: The identifier of the symbol
- **Type**: The type of the symbol (Alias, Equation, Parameter, Set, or Variable)
- **Dim**: The dimension of the symbol
- **Records**: The number of records of the symbol
- **Text**: The explanatory text of the symbol
A specific column can be sorted by clicking on the respective column header. Clicking again toggles the sorting direction. The Filter facility provides dynamic filtering of the list of symbols contained in the GDX file. By default, only the Name column is taken into account by the filter. The right side of the GDX Viewer displays the actual data of the symbol that is selected in the symbol table. Data can be displayed in either the List View or the Table View which can be toggled using the corresponding button in the upper part of the GDX Viewer.

The GDX Viewer automatically saves and restores its state as soon as data is reloaded due to changes to the underlying GDX file. Changes to the symbol table like an applied filter or the currently selected symbol are kept. Individual symbols keep applied filters as well as the view settings (e.g. the currently selected view (List View or Table View), preferences and the visible attributes for variables and equations) as long as the name, type, and dimension of the symbol did not change during the update of the GDX file. Resetting the state of a symbol can be achieved by clicking on the Reset button. In order to get rid of the whole GDX Viewer state, it has to be closed and reopened.

6.13.2.4.1 List View  The List View is the default representation when looking at data using the GDX Viewer. The data is presented in the form of a table in which each row represents a record of the symbol. Each record consists of key columns corresponding to the symbols dimension. The value columns vary according to symbol type:

- **Set/Alias**: Column Text contains the explanatory text of a record, Y if no explanatory text is available.
- **Parameter**: Column Value contains the numerical value of a record.
- **Variable/Equation**: Five numerical columns: (Level, Marginal, Lower, Upper, Scale).
The header displays the individual domains as well as the original index position using superscript notation which is especially useful if the column order has been changed. Furthermore, it allows to manipulate the displayed data in several ways. By left clicking on a column header, the corresponding column is sorted using a stable sort mechanism that does not change the order of equal entries. Clicking again changes the sorting direction. Due to the Extended Range Arithmetic used by GAMS, columns containing numerical data can contain special values. Those values are treated in a special way when being sorted:

- **-INF**: Smallest numerical value.
- **EPS**: Treated as value very close but different from 0.
- **+INF**: Largest numerical value.
- **NA**: Treated as first non-numerical value (sorted as being greater than +INF).
- **UNDEF**: Sorted as being greater than NA.
- **ACRONYMS**: Sorted as being greater than UNDEF. Relative order of acronyms is by their internal number.

The order of columns can be changed by dragging a column header and dropping it at a different position. Dragging the border between column headers allows for resizing the widths of the involved columns. The columns in the List View offer a filter facility which can be opened either by right clicking on the column header or by left clicking on the filter icon. The Reset button in the top right corner can be used to reset the view to defaults. This removes all filters, resets the applied sorting as well as the applied numerical format and switches back to the List View in case the Table View was active.

The button Preferences opens a menu containing different settings for the current symbol. Squeeze Defaults can be used on variables and equations in order to hide all columns that have the default value of the respective variable or equation type. Squeeze Trailing Zeros allows to turn on/off the truncation of trailing zeroes. The displaying format of numerical values can be specified using the Format drop down menu and the Precision spin box.

The available options for the Format are as follows:

- **g-format**: The display format is chosen automatically: **f-format** for numbers closer to one and **e-format** otherwise. The value in the Precision spin box specifies the number of significant digits. When precision is set to Full, the number of digits used is the least possible such that the displayed value would convert back to the value stored in GDX. Trailing zeros do not exist when precision=Full.
- **f-format**: Values are displayed in fixed format as long as appropriate. Large numbers are still displayed in scientific format. The value in the Precision spin box specifies the number of decimals.

- **e-format**: Values are displayed in scientific format. The value in the Precision spin box specifies the number of significant digits. When precision is set to Full, the number of digits used is the least possible such that the displayed value would convert back to the value stored in GDX. Trailing zeros do not exist when precision=Full.

---

![Figure 6.6 Variable mix from the prodmix model using g-format with precision=6](image1)

![Figure 6.7 Variable mix from the prodmix model using g-format with precision=Full](image2)

![Figure 6.8 Variable mix from the prodmix model using f-format with precision=6](image3)

![Figure 6.9 Variable mix from the prodmix model using e-format with precision=6](image4)
The value in the Precision spin box specifies the number of decimals or the number of significant digits depending on the chosen format:

- **g-format**: Significant digits [1..17, Full]
- **f-format**: Decimals [0..14]
- **e-format**: Significant digits [1..17, Full]

Next to Preferences is the Attributes button. This button opens a menu that allows to hide certain attribute columns of variables and equations explicitly. This is especially useful when using the Table View.

### 6.13.2.4.2 Table View

Switching to the Table View is possible for symbols that have at least two dimensions. This mode consists of two parts: A separate header view that shows the domains and attributes of the symbol as well as the actual data. The header view also allows to access the filter facility and highlights the positions of specific domains by using visual separators (from left to right: rows, columns, attributes). The data itself is reshaped into a format where the labels of the last dimension are moved into the column header while the remaining dimensions go into the row header. The entries in the table are the numerical values (or explanatory text in case of GAMS sets) of the record with the corresponding labels of the headers. The following image shows the List View of a GAMS Parameter on the left side and its representation in the Table View on the right side:

![Figure 6.10 Two dimensional parameter as List View and Table View](image)

In case of GAMS variables and equations, a virtual dimension for the five numerical values (Level, Marginal, Lower Bound, Upper Bound, Scale) is introduced and displayed in the column header. When looking at the data of variables or equations it is most of the time useful to visualize only a specific attribute like the variable levels. This can be achieved by using the Attributes menu in the upper part of the GDX Viewer. The menu allows to hide specific attributes explicitly by disabling the corresponding checkboxes.

Both column and row header dimensions can be moved into different positions in order to reshape the table view. This can be done by performing a drag-and-drop operation that starts by clicking (and holding) the left mouse button while the mouse cursor is located on the index that should be moved. Moving the mouse to an arbitrary location will bring up a drop indicator (in the form of a thin line) that previews the position in which the index will be dropped as soon as the mouse button is released. In general all index positions can be moved to arbitrary new locations, but there are a few restrictions:

- The (virtual) numerical dimension of variables and equations (Level, Marginal, Lower Bound, Upper Bound, Scale) can not be moved and is always the last dimension in the column header.
If a header (column or row) runs out of dimensions, a dummy header (Value, Text) is introduced which can not be moved but can be used as a drop target for further drag-and-drop operations.

<table>
<thead>
<tr>
<th>s</th>
<th>t</th>
<th>z</th>
<th>m</th>
<th>w</th>
<th>c</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jan</td>
<td>standard</td>
<td>rab-fod</td>
<td>gram</td>
<td>mus-rap</td>
<td>sc-mill</td>
<td>sc-gur</td>
</tr>
<tr>
<td>nwfp</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.19</td>
<td>0.19</td>
<td>0.22</td>
</tr>
<tr>
<td>pmw</td>
<td>0.2</td>
<td>0.05</td>
<td>0.2</td>
<td>0.14</td>
<td>0.14</td>
<td>0.22</td>
</tr>
<tr>
<td>pcw</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.09</td>
<td>0.09</td>
<td>0.23</td>
</tr>
<tr>
<td>ps</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.09</td>
<td>0.09</td>
<td>0.23</td>
</tr>
<tr>
<td>scwn</td>
<td>0.21</td>
<td>0.11</td>
<td>0.21</td>
<td>0.14</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>scwn</td>
<td>0.21</td>
<td>0.11</td>
<td>0.21</td>
<td>0.14</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>scw</td>
<td>0.27</td>
<td>0.11</td>
<td>0.21</td>
<td>0.14</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>srw</td>
<td>0.27</td>
<td>0.11</td>
<td>0.21</td>
<td>0.14</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>nwfp</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.19</td>
<td>0.19</td>
<td>0.22</td>
</tr>
<tr>
<td>pmw</td>
<td>0.2</td>
<td>0.05</td>
<td>0.2</td>
<td>0.14</td>
<td>0.14</td>
<td>0.22</td>
</tr>
<tr>
<td>ps</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.09</td>
<td>0.09</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Figure 6.11 The table view representation of a GAMS parameter during a drag-and-drop operation that will move the dragged dimension into the position indicated by the drop indicator (thin black line)

While filters that have been applied in the List View will also be applied to the Table View and vice versa, sorting does not have any effect. The labels are sorted using the internal order which can not be changed by the user. For large data it is recommended to reduce the amount of visible records by applying filters in the List View and by disabling certain value columns in case of variables and equations. Once the data has been reduced, switching to the Table View allows to look at the data in a compact format.

6.13.2.4.3 Filter Facility Filters can be applied in both the List View and the Table View through the header view that contains the domains as well as the (numerical) attributes. In the List View, the header is part of the data view itself, while the Table View offers a separate header view above the actual data. The filter dialog can be opened either by right clicking on the header or by left clicking on the filter icon. Depending on the type of the column (key or value), the appropriate filter dialog for labels or numerical values is opened.

The filter dialog for a specific key column shows all occurring labels of that column in a list. Per default all labels in all columns are visible. Shrinking the displayed data can be achieved by unchecking one or more labels. The filter is applied by clicking the Apply button. Instead of selecting labels manually one can also use the filter in the upper part of the filter dialog. More about the filter can be found in the Filter Section. The filter is automatically applied to the list of labels. Matching labels will be selected while all others will be deselected. The Hide unselected items checkbox can be used to automatically hide all deselected labels. This is especially useful for larger amounts of labels in combination with the filter as only the matching labels will remain in the list while typing. The Select All, Invert, and Deselect All buttons speed up getting the desired selection.
Numerical columns offer a different filter type that allows to specify an interval for values that should be included or excluded in the view for this specific column. The Min value is the lowest and the Max value is the largest one still included in the view. When the Exclude check box is activated, the specified range is excluded from the view instead of included. The lower part of the filter contains check boxes for special values like +INF or EPS. Checking/unchecking will show/hide the corresponding values in the view. Clicking the Apply button adds the filter to the column and refreshes the view accordingly. The Reset button removes an active filter.
Note

When a filter on a numerical column is applied, the values used for comparison are the actual values from the GDX file. Since the displayed values are influenced by the Format and the Precision setting, they can differ from the internal ones from the GDX file. Here is an example: Let's assume we have a parameter p(i) with p("i1")=1.34 and p("i2")=2. With format set to g-format and precision set to 2, the values are displayed as 1.3 and 2. When applying a filter with the bounds (Min, Max) set to 1.33 and 2 both records are still shown even though 1.33 is greater than 1.3 (number shown in the GDX Viewer) since the internal GDX value used for comparison is actually 1.34.

Figure 6.13 Filter dialog on numerical column Value for parameter c in transport.gdx

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump to Symbol Search</td>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Moves the focus to the Symbol Search input field located above the symbol table</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl - A</td>
<td>Command - A</td>
<td>Selects all data (right side only)</td>
</tr>
<tr>
<td>Copy (comma-separated)</td>
<td>Ctrl - C</td>
<td>Command - C</td>
<td>Copy selection to clipboard using comma as separator</td>
</tr>
<tr>
<td>Copy (tab-separated)</td>
<td>Ctrl - Shift - C</td>
<td>Command - Shift - C</td>
<td>Copy selection to clipboard using tab as separator (right side only)</td>
</tr>
<tr>
<td>Copy Without Labels (comma-separated)</td>
<td>context menu only</td>
<td>context menu only</td>
<td>Copy selection without labels to clipboard using comma as separator (Table View only)</td>
</tr>
<tr>
<td>Copy Without Labels (tab-separated)</td>
<td>context menu only</td>
<td>context menu only</td>
<td>Copy selection without labels to clipboard using tab as separator (Table View only)</td>
</tr>
<tr>
<td>Auto Resize Columns</td>
<td>Ctrl - R</td>
<td>Command - R</td>
<td>Resizes all columns to content (right side only)</td>
</tr>
</tbody>
</table>
6.13 GAMS Studio

6.13.2.4.4 Summary of Actions and Shortcuts

6.13.2.5 Reference File Viewer

The Reference File Viewer is a useful tool to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS model and is created using the rf parameter when running the model.

![Figure 6.14 Parameter to create a symbol reference file when running the model](image)

When the model is compiled, the reference file with ".ref" extension is created and can be opened either using the menu: File > Open or double-clicking the "RefFile" entry in the Process Log:

![Figure 6.15 Process Log showing clickable entry for the reference file](image)

The Reference Viewer is categorized by a number of tabs:

- **All Symbols** shows an alphabetical listing of all symbols used in the model.
- **Set, Acronym, Variable, Parameter, Equation, and Model** shows an alphabetical listing of symbols of the same type. See Data Types and Definitions for more details on GAMS data types.
- **File** shows an alphabetical listing of all file statements used in the model.
• **Function** shows a list of GAMS built-in functions that has been used in the model.

• **Unused** shows an alphabetical listing of all symbols that have been declared in the model but are not used anywhere.

• **File used** shows an alphabetical listing of all files used in the model including the full file path. A double-click on an file entry of the table will jump to the beginning of the reference file.

The number next to the tab name indicates how many symbols in the category, where the number of All Symbols tab sums up the number of symbols in all other tabs except Unused tab and File used tab. A click on a symbol entry of the table in every tab (except File used tab) will show the detailed reference list containing the reference location in the right hand side of the reference viewer. A mouse-over on an entry of the reference list will show the tooltip of file location and position with the file that has been referenced. A double-click on an entry of the reference list will jump to the position in the corresponding file that has been referenced.

The reference list is organized by the following reference types:

<table>
<thead>
<tr>
<th>Reference Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declared</td>
<td>Declaration of the symbol.</td>
</tr>
<tr>
<td>defined</td>
<td>Defined using a data statement.</td>
</tr>
<tr>
<td>assigned</td>
<td>When the symbol appears on the left side of an assignment statement.</td>
</tr>
<tr>
<td>implicitly assigned</td>
<td>Implicit assignment like a variable in a model.</td>
</tr>
<tr>
<td>control</td>
<td>When the set is used as a control set.</td>
</tr>
<tr>
<td>reference</td>
<td>Referenced in a statement.</td>
</tr>
</tbody>
</table>

The detailed description of the reference types and their shorthand symbols can be found in Reference Types in GAMS Output.

### 6.13.2.5.1 Sort and Filter referenced symbols

- **Sort** : A click on a table column header in the reference table will sort the symbol in the table by either Entry, Name, Type, Dim(ension), Domain, or Text.

- **Filter** : An input in the Filter box will filter the symbols in the reference table by Name. More about the filter can be found in the Filter Section.

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol Filter</td>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Jump to the Symbol Filter input field located above the symbol table</td>
</tr>
<tr>
<td>Auto Resize Columns</td>
<td>Ctrl - R</td>
<td>Command - R</td>
<td>Resize all columns to content (right side only)</td>
</tr>
</tbody>
</table>

### 6.13.2.5.2 Summary of Actions and Shortcuts

### 6.13.2.6 Solver Option Editor

The Solver Option Editor is used to view and edit a solver-specific option file for controlling solver and interpreting results. See usage of The Solver Options File in Solver Usage and how to set optFile parameter to instruct gams to read an option file in The GAMS Call and Command Line Parameters.
The Solver Option Editor provides an editor to edit the file contents in a table form where a row entry corresponds to a line in the solver option file. The Solver Option Editor also provides a browser of solver option definitions which can be displayed in group or be filtered by filter-term and where the option can be directly added into or deleted from the editor.

The Solver Option Editor consists of the contents editor in the left pane, option definition browser in the right pane, as well as message and configuration tab in the bottom pane.

- The left pane shows the contents editor of solver option file in a table form. Each row entry is either an option entry containing the option key, option value, as well as end-of-line comment (shown if there is definition of end-of-line comment characters for a solver option file) or a comment entry. Below the editor shows Compact View checkbox and Open As Text button
  - An option key and value are shown in green color when defined correctly according to the definition, otherwise in red color when there is an error. An end-of-line comment (if defined) and a comment entry are shown in grey color. (see section Toggle comment/option selection on how to turn an option entry into a comment entry and vice versa)
  - A comment row entry is shown in a merged cell as differences between option key van value are not identified in a comment line of a solver option file.
  - The Compact View checkbox enables the editor to display the file contents without comments. Note that some editing actions are suppressed when the editor is in compact view (see section Compact View for more details).
  - The Show Messages checkbox allows to display or hide the messages log at the bottom pane.
  - The Open As Text button allows to reopen the file in the text editor once contents of the file has been saved (see Open an existing solver option file and Save a solver option file in section Basic Operations for more details).

- The right pane shows a browser of solver option definitions.
  - The lower part of the browser lists all option definitions, each contains Option Name, Synonym, Default Value, Range, Type, and Description. The list can be sorted alphabetically in ascending or descending order by clicking the Option Name header. A checkbox in front of an option entry indicates whether there is an option of this definition entry defined in the left pane editor. An entry with enumeration type (EnumStr or EnumInt) can be expanded to show all
enumerated values of the option by clicking the bullet in front of the option entry, clicking the bullet again hides all enumerated values of the option. By double clicking the entry or dragging the entry and dropping in the left pane editor adds the option key with the default value defined by this option definition in the left pane editor (See section Summary of Actions and Shortcuts on how to add or insert an option).

- Above the list of all option definitions is a group of options and a filter box. Option groups filter the list of option definitions by group name. By default all option definitions are displayed. Filter box with placeholder text Filter Options ... allows to filter the option definitions. As a filter term has been typed into the filter box the results are displayed in the list of option definitions below. More about the filter can be found in the Filter Section.

- The bottom pane shows the message log which reports the operations that have been carried out such as the contents have been loaded and saved as well as errors resulting from editing the contents. By default the message log is shown unless the Show Messages checkbox in the left pane has been unmarked to hide the message log.

6.13.2.6.1 Basic Operations

- **Create a solver option file**: via either the Studio menu or the project context menu in project explorer.
  - From the Studio menu, choose File > New... then enter a valid solver option file name and suffix into the file dialog. choosing Add new file by right clicking on project name in project explorer brings up the same file dialog for entering a valid solver option file name and suffix (See how to set different suffix values for solver option file from optFile parameter).
  - From the project context menu, right clicking on project name in project explorer and choosing Add new solver option file brings up a list of solver names. Selecting a solver name from the list brings up a file dialog with the selected solver name as file name and the default option file suffix name opt (See how to set different suffix values for solver option file from optFile parameter).

- **Open an existing solver option file** : via either the Studio menu or project context menu in project explorer.
  - From the Studio menu, choose File > Open... and select Option Files or All Files in the file dialog and then select a solver option file.
  - From the project context menu, right click on project name then choose Add Existing file then choose Option Files or All Files in the open file dialog and then select a solver option file.
  - In case a solver option file is already opened with solver option editor, the file can be reopened in text editor by choosing Reopen File As Text from the file context menu in project explorer.
  - It is possible to open the solver option file in either solver option editor or text editor.
    - In case a solver option file has already been opened in solver option editor, choosing Reopen File As Text from the file context menu in project explorer closes the solver option editor and reopens the file in the text editor. Note that Open As Text button in the left pane editor performs the same function.
    - In case a solver option file is already opened in text editor, choosing Reopen File using Solver Option Editor from the file context menu in project explorer closes the text editor and reopens the file in the solver option editor.
    - In case a file is already listed under the project entry but not yet opened, choosing Open File from the project context menu opens the file in the solver option editor; however, choosing Open File As Text from the project context menu opens the file in the text editor.

- **Save a solver option file**: activated by choosing either File > Save menu or Ctrl - S shortcut. Choosing File > Save As menu or Ctrl - S shortcut brings up a file dialog to choose a file with different name to be saved as (See also how to set different suffix values for solver option file from optFile parameter).
6.13.2.6.2 Navigating the contents

When a component of the solver option editor in focus, it is possible to perform further actions (see section Summary of Actions and Shortcuts for the list of actions and shortcuts). Clicking the component area will bring the component into focus or in selection. In addition to using mouse to navigate the solver option editor components, it is also possible using keyboard. Press Tab key navigates the components of the solver option editor. For example, when left pane editor is in focus, pressing Tab navigates from the left pane editor, to the right pane option group box, to the right pane Filter box, and to the right pane definition browser in the describing order. When the left pane editor is in focus, press Up, Down, Left, and Right keys to navigate the left pane editor table. When the right pane browser is in focus, press Up and Down to navigate the definition entries, press Right to expand/show the enumerated entries of the definition, and press Left to collapse/hide the enumerated entries of the definition.

There is a connection between an option entry in the left pane editor and a definition entry in the right pane browser. When navigating the solver option editor, it is possible to identify this connection from both the left pane editor and the right pane browser.

- From the left pane editor, right click on the selected entry and choose show option definition from the context menu. The entry in the right pane browser that contains the definition of the selected entry is highlighted and selected. When a cell or a row is already selected the shortcut Ctrl - F1 delivers the same behavior.

- From the right pane browser, click or select on an entry. All entries in the left pane editor will be highlighted and selected if the definition has already been added. Otherwise there will be no selection in the left pane editor.

More editing actions can be performed on both the left pane editor and the right pane browser (See Editing the Contents for details).
6.13.2.6.3 Editing the Contents  The followings describe actions that can be performed when editing the contents:

- **Edit option key, value, and comment**: This action performs in the left pane editor.
  - when the cell in the left pane editor is selected, double click on the cell or press platform-dependent edit key (e.g., F2) in order to edit option key, value, or comment (if available). A drop-down list suggests possible option keys and option values as when possible. Press **Enter** to confirm the edit and press **Esc** to cancel the edit.

- **Add new option**: This action appends a new option entry as the last entry of the left pane editor. There are several ways to add a new option entry.
  - click on the plus icon in the header of the left pane editor, a new option entry with dummy option key [KEY], option value [VALUE], and option comment [comment] (if available) is appended as the last entry.
- it is also possible to add new option from the the right pane browser. Select the definition entry in the right pane browser that has not been added/inserted into the left pane editor (a solid checkbox in front of the entry is not marked), double click or right click on the selection then choose add this option. A new option entry ith the option key and default value of this definition will be added as the last entry in the left pane editor, and dummy option value [VALUE] is appended as the last entry.

- **Insert new option and comment**: It is possible to perform this action from the left pane editor and from the the right pane browser and there are several ways to insert a new option or a comment entry.

  - right click on the selected entry in the left pane editor and choose insert new option from the context menu. A new option entry with dummy option key [KEY], option value [VALUE], and option comment [comment] (if available) will be inserted before the selected entry. When a cell or a row is already selected the shortcut Ctrl - Return delivers the same behavior.

  - right click on the selected entry in left pane editor and choose insert new comment from the context menu. A new option entry with dummy text [COMMENT] will be inserted before the selected entry. When a cell or a row is already selected the shortcut Ctrl - Shift - Return delivers the same behavior.

- double click on a definition entry in right pane browser.

  - In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be added as the last entry in the left pane editor. When there is a selection on a cell or a row the shortcut Return delivers the same behavior.

  - In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to replace existing entry, or to add new entry, or to abort. Replace existing entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. Add new entry will add a new option entry with the option key and default value of this definition into the left pane editor. Abort will cancel the action. See section override existing option on how to suppress this default behavior.

When double clicking an enumerated value entry of the definition a new option entry will be added with the option key of the parent entry and the selected enumerated value.

- drag a definition entry from the right pane browser and drop in the left pane editor.

  - In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be dropped before the position of selected entry in the left pane editor.

  - In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to replace existing entry, or to add new entry, or to abort. Replace existing entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. Add new entry will insert a new option entry with the option key and value of this definition before the position of selected entry in the left pane editor. Abort will cancel the action. See section override existing option on how to suppress this default behavior.

See also section Settings on how to insert a new option together with a comment from the definition.

- **Toggle comment/option**: This action performs in the left pane editor, turning an option entry into a comment entry and turning a comment entry into an option entry.
select the option entry and right click on the selection, then choose toggle comment/option selection from the context menu. In case of an option entry this action turns the entry into a comment entry (shown in grey color). In case of a comment entry this action turns the entry into an option entry entry (shown in either green or red color depending on whether or not there is an error). When a cell or a row is already selected the shortcut Ctrl - * delivers the same behavior. Clicking on the color box in front of the entry row also delivers the same behavior, even without a selection.

- **Move up and down:** These two actions perform in the left pane editor, changing the order of the option and comment entries.

  - to move an entry up in the left pane editor: select the option entry and right click on the selection, then choose move up from the context menu. The selected option entry will be moved one position up the option entry table. When a cell or a row is already selected the shortcut Ctrl - Up delivers the same behavior.

  - to move an entry up in the left pane editor: select the option entry and right click on the selection, then choose move down from the context menu. The selected option entry will be moved one position down the option entry table. When a cell or a row is already selected the shortcut Ctrl - Down delivers the same behavior.

- **Deleted option:** It is possible to perform this action from the left pane editor and from the the right pane browser.

  - from the left pane editor, select the option entry and right click on the selected, then choose delete selection. The selected option entry will be deleted from the option entry table. When there is a selection on a cell or a row the shortcut Ctrl - Delete delivers the same behavior.

  - from the right pane browser, select the definition entry that has already been add/inserted into the left pane editor (a checkbox in front of the entry is marked) and right click on the selection then choose remove this option. All entries in the left pane editor defined by this definition will be deleted from the option entry table. When there is already a selection on a definition row the shortcut Delete delivers the same behavior.

- **Show option definition:** This action performs in the left pane editor. Right click on the left pane of the option editor and choose Show option definition. The option definition entry on the right pane will be highlighted. The shortcut Ctrl - F1 delivers the same behavior.

- **Show all options of the same definition:** This action performs in the left pane editor. Right click on the left pane of the extended editor and choose Show all options of the same definition. All option entries on the left pane will be highlighted. The shortcut Shift - F1 delivers the same behavior.

Figure 6.19 All entries of the same option are highlighted
6.13.2.6.4 Compact View  Compact View of the solver option editor allows to view and edit solver option without comments. Comments in a solver option file are not interpreted by either GAMS or the solver but used for documentation purpose. As the contents of the solver option file grows larger with several comment lines, it can be difficult to see which options eventually will be interpreted. To this end, the compact view becomes useful to hide all comments and show only non-comment lines. In the right pane editor, mark Compact View checkbox to activate the compact view and unmark the checkbox to deactivate the compact view. The comment entry is shown again once the compact view is deactivated.

Note that the result of some actions are not visible when in compact view. For example, action toggle comment/option selection when performed on an option row entry, turning the entry into a comment entry and therefore hidden from the compact view. Action delete selection when performed on an option row entry, removing the content entry and therefore no longer visible hidden from the compact view.

Some editing actions that can change the order of the contents are suppressed. These actions are insert new option, insert new comment, move up, and move down. Nevertheless adding or inserting option from the right pane are allowed, but a comment that has been inserted with the option (if set) are not visible in compact view. See also section Settings on how to insert a new option together with a comment from the definition.

6.13.2.6.5 Settings  The setting tab allows to configure the behavior when inserting new option from definition and deleting option. The setting can be accessed by opening the Settings dialog (File > Settings) and switching to the Misc. tab. These configurations are:

- **Override existing option.** This behavior allows Studio to override existing option when an option entry has been added or inserted from the right pane browser. In case there are more than one entry of the same option, Studio will pop-up a message box to offer three alternatives: either to replace existing entry, or to add new entry, or to abort.
  
  - Replace existing entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition.
  
  - Add new entry will add or insert a new option entry with the option key and default value of this definition into the left pane editor.
  
  - Abort will cancel the action.

  By default this behavior has been set. In the Settings dialog, unmark the checkbox in front of the text overriding existing option to suppress the behavior and mark to enable the behavior.

- **Add option description as comment above.** This behavior allows Studio to add option description as additional comment entry above an option entry that has been added or inserted from the right pane browser. The description has been taken from the option definition in the right pane browser. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text add option description as comment above to enable the behavior and unmark the checkbox to suppress the behavior.

- **add option description as end of line comment** This behavior allows Studio to add option description as additional end-of-line comment of an option entry that has been added or inserted from the right pane browser. The end-of-line comment is only available when the solver defines valid end of line characters. The end-of-line comment column will be shown in the solver option editor only when available (there is definition of end-of-line comment characters for a solver option file). By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text add option description as end of line comment to enable this behavior and unmark the checkbox to suppress the behavior.

- **delete all immediate comments above** This behavior allows Studio to delete all immediate comments (if there is any) above when deleting an option. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text delete all immediate comments above to enable this behavior and unmark the checkbox to suppress the behavior.
6.13.2.6 Summary of Actions and Shortcuts

Actions and their shortcuts that can be performed via the left pane of solver option editor are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle option/comment selection</td>
<td>Ctrl - <code>*</code></td>
<td>Command - <code>*</code></td>
<td>Toggle between option and comment</td>
</tr>
<tr>
<td>Insert new option</td>
<td>Ctrl - Return</td>
<td>Command - Return</td>
<td>Insert a new option</td>
</tr>
<tr>
<td>Insert new comment</td>
<td>Ctrl - Shift - Return</td>
<td>Ctrl - Shift - Return</td>
<td>Insert a new comment</td>
</tr>
<tr>
<td>Delete selection</td>
<td>Ctrl - Delete</td>
<td></td>
<td>Delete the selected option/comment</td>
</tr>
<tr>
<td>Move up</td>
<td>Ctrl - Up</td>
<td>Command - Up</td>
<td>Move the selected option/comment up for 1 row</td>
</tr>
<tr>
<td>Move down</td>
<td>Ctrl - Down</td>
<td>Command - Down</td>
<td>Move the selected option/comment down for 1 row</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl - A</td>
<td>Command - A</td>
<td>Select all options</td>
</tr>
<tr>
<td>Show option definition</td>
<td>Ctrl - F1</td>
<td>Command - F1</td>
<td>Show definition of this selected option in the right pane</td>
</tr>
<tr>
<td>Show all options of the same definition</td>
<td>Shift - F1</td>
<td></td>
<td>Show all options of the same definition defined in the right pane</td>
</tr>
<tr>
<td>Resize columns to contents</td>
<td>Ctrl - R</td>
<td>Command - R</td>
<td>Resize the columns in the left pane to contents</td>
</tr>
</tbody>
</table>

Actions and their shortcuts that can be performed via the right pane of solver option editor are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Filter</td>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Jump to the focus to the Option Filter input field</td>
</tr>
<tr>
<td>Add this option</td>
<td>Return</td>
<td>Return</td>
<td>Add option in the left pane from the selected definition</td>
</tr>
<tr>
<td>Remove this option</td>
<td>Delete</td>
<td></td>
<td>Remove option defined by this definition from the left pane</td>
</tr>
<tr>
<td>Copy option name</td>
<td>Ctrl - C</td>
<td>Command - C</td>
<td>Copy option key from this selected definition</td>
</tr>
<tr>
<td>Copy option description</td>
<td>Shift - C</td>
<td>Shift - C</td>
<td>Copy option description from this selected definition</td>
</tr>
<tr>
<td>Copy definition text</td>
<td>Ctrl - Shift - C</td>
<td>Command - Shift - C</td>
<td>Copy option text from this selected definition</td>
</tr>
<tr>
<td>Resize columns to contents</td>
<td>Ctrl - R</td>
<td>Command - R</td>
<td>Resize the columns in the right pane to contents</td>
</tr>
</tbody>
</table>

6.13.2.7 GAMS Configuration Editor

The GAMS Configuration Editor is used to view and edit a GAMS Configuration File in YAML Format. The editor shows the two configuration sections: command line parameters (commandLineParameters), and operating system environment variables (environmentVariables), in different tabs of the editor.
Note that GAMS Configuration editor does not show the contents of the third section: external solver configuration (solverConfig) if exists. However, the contents in this section will be written out to the file when saving the file via the editor. It is possible to use Reopen File as Text from Project Explorer context menu to reopen and edit the file contents in text editor when the file is already opened in the GAMS Configuration Editor and use Open File as Text from Project Explorer context menu to open the file in text editor when the file is not yet opened in any editor. See the syntax of gamsconfig.yaml file from GAMS Configuration File in YAML Format.

GAMS will process a sequence of gamsconfig.yaml files from different locations (see the location of GAMS Configuration file from The GAMS Configuration in YAML Format and the order of precedence for command line parameters from Order of Precedence for Options). Studio can create a new gamsconfig.yaml file at the default user-specific location and open the file in the editor when the file at the location does not exist. This can be done by choosing GAMS > Default GAMS Configuration. In case the file already exists at the default location, choosing GAMS > Default GAMS Configuration will not override the gamsconfig.yaml in the default user-specific location but open the file in the editor.
It is possible to save the `gamsconfig.yaml` file into different location using `File > Save As...`

### 6.13.2.8 Tab Browser

The Tab Browser is accessible with a button next to the main tab bar or via the shortcut `Ctrl - 4`. It opens a list of all open files and features a search input field. When opening the tab browser, the search field is already focused. Typing updates the list in real time. The `Enter` key selects the first item in the list and opens it. It is also possible to use arrow up/down to navigate the list and select an item with `Enter`. The filter supports wildcard syntax.

### 6.13.2.9 Pin View

Studio provides a Pin View to carry a clone of a selected tab. This allows to have two file contents visible at the same time. To pin a tab there are two entries in the tabbar: "Pin right" (shortcut `Ctrl - Click`) splits the edit area horizontally and opens the Pin View right to the main tabs. "Pin below" (shortcut `Ctrl - Shift - Click`) splits the edit area vertically and opens the Pin View below the main tabs. The state and size of the Pin View is restored on restarting Studio.
The Pin View provides additional buttons. The first button switches the split orientation from right to below and vice versa. The second button allows to activate synchronized scrolling for the two visible editors. Synchronizing is always passed from the active editor containing the focus to the passive editor, which means that scrolling in the passive editor with the mouse wheel isn’t passed back to the active editor allowing adjustments without switching the function off and on. The last button closes the Pin View.

### 6.13.2.10 Navigation History

The Navigation History records a navigation history event each time the cursor is moved in some kind of text file opened in the Studio central widget. This allows the user to quickly jump back (and forward again) to text passages that were recently edited. To reduce clutter some events are set to overwrite the previous event. This happens when the cursor is moved by only a single position in either direction, so that when a user jumps through text or types something only the latest position will be put into the history. The idea behind this is to make it easier and more convenient to jump to positions that actually matter. Using the navigation history is not limited to a single file. It is easy to jump between two or more files to go back and review recent changes. If a file was closed in the meantime it will be reopened when navigating through the history. This includes even files that were completely removed from Studio. When such a file is reopened through the Navigation History feature, due to technical limitations, the affiliation of the file to its previous project is lost. Please note that after navigating back in the history all "future" history events will be deleted and overwritten if the user creates new history events.

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Alternative Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go Back</td>
<td>Alt - Left Arrow</td>
<td>Ctrl - [</td>
<td>Mouse Button 4</td>
</tr>
<tr>
<td>Go Forward</td>
<td>Alt - Right Arrow</td>
<td>Ctrl - ]</td>
<td>Mouse Button 5</td>
</tr>
</tbody>
</table>
6.13.3 Further Studio Widgets

6.13.3.1 Project Explorer

The Project Explorer organizes the files opened in Studio into projects. Each project has its own base directory. All files associated to the project are organized in a hierarchical tree structure of folders relative to this base directory. A project's tooltip tells the location of the project in the file system. To support faster detection there are several icons for different file types:

- The runnable GAMS files (currently supported .gms and .inc) are listed with GAMS icon,
- the main GAMS file assigned to the project additionally contains a green play symbol,
- GDX files (.gdx) have an database icon,
- Reference files (.ref) are shown with a reference nodes on a sheet,
- editable files (e.g. .txt) contain a pen on a sheet,
- read-only files (e.g. .log) contain lines on a sheet.

The tooltip of a file or folder entry tells the location in the file system. The file location, the project location, and the folder location can be explored using the default file system explorer via the context menu "Open location". Selecting "Open terminal" opens the system's terminal in the location of the selected file, project, or folder. The context menu of a .gdx file provides a special entry "Open in GDX Diff" that opens the specific GDX Diff Dialog.
An empty project can be created by selecting New Project from the File menu or the context menu in the Project Explorer. A file dialog opens to select the project folder. The project gets the name of that folder. If a project with that name already exists, Studio looks for a number to append until the projects name is unique. Additionally a project is created automatically when opening a file. The projects name and location is determined by the name and location of the opened file. One project may contain multiple runnable GAMS files but only one runnable GAMS file is set as the main file, marked with a green arrow overlay. By default the first runnable GAMS file in a project is considered the main file. The tooltip of a project also tells the location of the folder and the name of the current main file. Additionally the name of the recently created output file (.lst) is shown after the first run. The main file can be changed to other runnable GAMS file in a project via the context menu "Set as main file". The project base directory remains unchanged if the runnable GAMS file is changed.

The main file of the active project will be executed when pressing F9 or clicking the execute button (see how to set GAMS parameters and execute the main file in Quick Access Toolbar and GAMS Parameter Editor). To better recognize the active project, the current file and its project are in bold text and the icons of other projects are slightly dimmed. After executing the main file, the generated listing file (.lst) is added to its project and the output view (see Process Log) is opened. A clickable log line in the process log may add the corresponding file into the project and open the file in the Central Widgets area. Users can manually add more files to a project by right-clicking and choosing either "Add existing file" or "Add new file" from the context menu. Additionally Drag and Drop from the systems file explorer is supported.

Files can also be removed from a project and projects can be removed from Project Explorer via the two actions "Close file" and "Close project" from the context menu. The action "Close project" closes the project and all files in the project if opened as well as removes all file entries and the project entry from the Project Explorer without actually touching the files on the file system. The action "Close files in this folder" closes all files in this folder and it's subfolders. The action "Close file" closes the file if opened and removes the file entry from the project. In case the closed file is the only entry in a folder, the folder is also closed accordingly. In contrast a project can be empty during the current session but an empty project isn't restored on Studio restart. In case there are unsaved changes a message popup will appear, asking the user how to proceed.

Next to each project there is a cog wheel button that opens the Project options page. It contains these actions:

- Name: defines the name of the project
• Working directory: defines the default directory where the main GAMS file associated to the project will be executed

• Base directory: defines the directory from where the folders to each file of the project are generated. Changing this will update the folder structure to all files of this project.

• Main GMS file: informs about the current main GMS file.

The changed Project Options can be applied by selecting File > Save or pressing Ctrl - S.

It is possible to select multiple items in the project explorer. A selection always contains items of one kind, files or projects never both depending on the first selected item. Folders in this case count as multiple file selection. The context menu then addresses all selected items.

A project can be 'exported' using the context menu to a 'GAMS Studio Project' with the suffix gsp. The files assigned to the project are referenced relatively to the projects base directory. Therefore it is highly recommended to export the project to the base directory itself to keep its consistency. The relative paths allows to share the project along with its other files. When a project file is 'imported' into GAMS Studio, the base directory is set to the location of the project file. Starting from this directory Studio expects the other files of this project.

Selected files can be dragged to another project. By dropping them they are moved to the destination project. If a folder is empty after this action its representation is removed from the project. Pressing the Ctrl key creates a copy of the file-reference in that project (no file copy). If the project already contains a specific file it is just ignored.

Actions like rename, move, or delete from file system are currently not supported but will be added in the future.

Note

For long-time users of the classic GAMS IDE the section about the differences between IDE projects and Studio projects might be of particular interest.

6.13.3.2 Process Log

After starting a GAMS process the LogViewer is opened to view the process Log. Usually the Log stays at the bottom to keep tracking the latest log lines. The tracking can be paused by clicking on the up arrow of the vertical scrollbar. To continue tracking drag the scrollbar to the bottom.

The Log usually contains links of different kinds:

• compilation errors (red): Links to the gms file where the error occurred. Additionally contains a link to the listing file (.lst) at the beginning of the line (blue).

• listing links: Links to a specific line in the listing file

• file link (green): Links to a file without line numbers such as .gdx or .ref

Clicking on a link takes the focus to that file (and line). Double-clicking in the Log looks for the nearest link to the listing file and takes the focus there. Files that were not part of the current project are added when they are opened by link.

For general messages there is the System Log stacked together with all Process Logs. It keeps track of Studio related messages not originated from a specific GAMS run.
6.13 GAMS Studio

6.13.3 Quick Access Toolbar and GAMS Parameter Editor

The toolbar contains two parts: a quick access section and GAMS Parameter editor to customize how GAMS is executed the model (see The GAMS Call and Command Line Parameters).

The quick access section contains icons for performing common File actions like New, Open, and Save, as well as accessing Settings dialog, Model Library Explorer dialog, Project Explorer view, Process Log, and Help view.

The GAMS Parameter Editor provides a way to control and customize the execution of a GAMS model. The editor displays the parameters of the project of the file that is currently opened in Central Widgets (see also Project Explorer for project and its runnable files). The parameters can be typed into a combobox and the execution of a GAMS model can be carried out using the following pre-customized execution commands:

- **Run**: Choose Run or press F9 to compile and execute GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the default parameter action=CE (Compile and Execution).

- **Run with GDX Creation**: Choose Run with GDX Creation or press F10 to compile and execute GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters action=CE and gdx=default.

- **Compile**: Choose Compile or press Shift - F9 to compile GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the parameter action=C (CompileOnly).

- **Compile with GDX Creation**: Choose Compile with GDX Creation or press Shift - F10 to compile GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters action=C and gdx=default.

- **Run NEOS**: Choose Run NEOS to compile GAMS statements in the main runnable file locally and pass the execution to the NEOS server.
- **Run GAMS Engine**: Choose Run GAMS Engine to compile GAMS statements in the main runnable file locally and pass the execution to the GAMS Engine server.

The pre-customized execution command will operate on the main file (marked with a little green arrow over the regular icon) of the active project (see how to manage the main file in section Project Explorer). In case GAMS parameters have been set they will be appended to the pre-customized commands. Just like the GAMS Terminal, when there are identical parameters with different values the last one "wins" and overwrites the previous ones. Thus it is possible for a user to change and override GAMS Studio default parameters, which can possibly lead to problems if done incorrectly. In this case Studio prints out a warning message, informing users about potential problems. To debug what Studio does the full GAMS call parameters will be printed to the System Log. After execution, the parameters will be added into the project history. All files in a project share one parameter history. The previous GAMS parameters can be recalled from the project history via the drop-down menu of the combobox. Switching the file opened in Central Widgets will activate the different project history only when the project of the newly opened file is different from the project of the file before the switch.

The parameters `curDir` and `workDir` behave slightly differently in Studio compared to the terminal. When using one of these parameters on the command line GAMS expects the gms file to be in the given folder. In Studio the path to the input file is always given as an absolute one. Therefore, there is no need to use the `inputDir` parameter to make the gms file accessible to GAMS if it is not in the specified working directory.

The GAMS Parameter Editor provides a way to either interrupt or stop the currently running job when an execution of a GAMS model is in progress (indicating with animated icon over the project folder icon in Project Explorer):

- **Interrupt**: Click the Interrupt button or press F12 to send an interrupt request to the running job in order to perform a graceful stop and collect an incumbent result back from the execution if the solver supports this feature. The command is enabled when there is a job that is currently running in the project and disabled when there is no currently running job in the project.

- **Stop**: Click the Stop button or press Shift - F12 to send a request to stop the running job immediately. The command is enabled when there is a job that is currently running in the project and disabled when there is no currently running job in the project.
The Extended GAMS Parameter Editor allows to configure the GAMS parameters. The extended parameter editor is shown when the Show button next the parameter combobox is clicked or with the shortcut Ctrl - Alt - 3. The editor can be hidden when the button is clicked again. When shown, the GAMS parameters from the parameter combobox will appear as a list of entries in the left pane of the extended parameter editor, each entry contains the Key and Value.

Note that the GAMS parameter combobox will be disabled when the extended parameter editor is shown and all editing has to be done in the extended parameter editor.

The right pane of the extended editor displays the list of all valid parameter definitions, each contains Parameter Name, Synonym, Default Value, Type, and Description. The parameter definition entry with enumeration type (EnumStr or EnumInt) can be expanded to show all enumerated values of the parameter by clicking the bullet in front of the parameter entry, clicking the bullet again will hide all enumerated values of the parameter. Above the list of all valid parameter definitions is a filter box (with placeholder text Filter Parameter...) to allow to filter all parameter definitions. As a filter term has been typed into the search box the results will be displayed in the list below. More about the filter can be found in the Filter Section.

The parameter editor displays potential parameter errors in red color both in the combobox and in the left pane of the extended editor. When hovering a mouse over the error key or value, a tooltip with more detailed explanation of the error appears.
Figure 6.26 Process Log reports parameter errors when running the model with an unknown option parameter

When there are several entries of the same parameter, all other entries except the last entry will be ignored when gams executes the model. In such case, the combobox of parameter editor displays a warning in yellow and the icon in front of the parameter in the extended editor turns half-yellow in combination of another color, red or green, depending of whether or not there is a potential error.

See edit parameter key and value on how to edit parameter key and value that contains an error or a warning. See show all parameters of the same definition on how to locate all entries of the same parameter.
The followings describes all editing actions that can be performed via the extended parameter editor:

- **Edit parameter key and value**: double click on key cell to edit **Key** and on value cell to edit **Value** of the entry. A drop-down list will suggest possible keys and values when possible. Press **Enter** to confirm the edit and press **Esc** to cancel the edit.
• **Add new parameter:** there are several ways to add a new parameter via extended parameter editor.
  
  – right click on the left pane of the parameter editor and choose **add new parameter** from the pop-up context menu
  
  – right click on the selected parameter entry of the parameter table and choose **add new parameter** from the pop-up context menu
  
  – click on the add new parameter button next to **Key** table header in the left pane
  
  A new entry will be added at the end of parameter entry table with dummy parameter key [KEY] and dummy parameter value [VALUE]. See **edit parameter key and value** on how to edit parameter key and value.

  It is also possible to add a new parameter from the right pane of the parameter editor by double clicking at the definition entry in the right pane. The selected definition entry will be added as a new parameter entry at the end of table in the left pane. In case of double clicking an enumerated value entry of a parameter definition the definition entry will be added as a new parameter with the selected enumerated value. Otherwise the default value of the entry will be added.

  ![Figure 6.31 Parameter with enumerated value added by double clicking at the selected entry in the right pane](image)

• **Insert new parameter:** right click on the selected entry and choose **insert new parameter**. A new entry will be inserted before the selected entry with dummy parameter key [KEY] and dummy parameter value [VALUE]. See **edit parameter key and value** on how to edit parameter key and value.

• **Move up:** right click on the selected entry and choose **Move Up**. The selected entry will be moved one position up the table. This action will change the order of parameters.

• **Move down:** right click on the selected entry and choose **Move Down**. The selected entry will be moved one position down the table. This action will change the order of parameters.

• **Delete selection:** right click on the selected entry and choose **Delete Selection**. The selected entry will be deleted from the table.

• **Delete all parameters:** right click on the left pane of the extended editor and choose **Delete all Parameters**. All entries will be deleted from the table.

• **Show parameter definition:** right click on the left pane of the extended editor and choose **Show parameter definition**. The parameter definition entry on the right pane will be highlighted.

• **Show all parameters of the same definition:** right click on the left pane of the extended editor and choose **Show all parameters of the same definition**. All parameter entries on the left pane will be highlighted.
Figure 6.32 All entries of the same parameter are highlighted

- **Resize columns to contents**: right click on the left or right pane of the extended editor and choose **Resize columns to contents**. All columns will be resized according to their contents.

- **Add this parameter**: right click on the right pane of the extended editor and choose **Add this parameter**. The parameter with default value (if defined) of the selected definition will be added as the last entry on the left pane of the extended parameter editor. This action in the context menu is enabled only when there is no entry of this parameter on the left pane. Note that a double click on the definition will also perform add this parameter action. Performing the action on the parameter definition that has already been added will pop-up a dialog indicating that the parameter exists on the left pane and prompting for one of the followings:
  - replace existing entry If there is only one entry or replace the first entry and delete other entries if there is more than one entry, or
  - add new entry of the same definition, or
  - show details of all entries of the same parameter, or
  - abort the action

Figure 6.33 A pop-up dialog when adding the definition parameter that has already been added

- **Remove this parameter**: right click on the right pane of the extended editor and choose **Remove this parameter**. The parameter of the selected definition will be deleted from the left pane of
the extended parameter editor. This action in the context menu is enabled only when there is an
entry of this parameter definition on the left pane. In case there are multiple entries of the same
parameter, all entries on the left pane will be deleted.

Press F1 on a parameter entry in the extended parameter editor will activate the The GAMS Call and Command Line Parameters help page containing the detailed description of the GAMS parameter.

### 6.13.3.3.1 Summary of Actions and Shortcuts

Actions and their shortcuts that can be performed via Toolbar and GAMS parameter editor are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Ctrl - N</td>
<td>Command - N</td>
<td>Open new file dialog</td>
</tr>
<tr>
<td>Open</td>
<td>Ctrl - O</td>
<td>Command - O</td>
<td>Open file open dialog</td>
</tr>
<tr>
<td>Save</td>
<td>Ctrl - S</td>
<td>Command - S</td>
<td>Save the current file. If the file does not exist on disk open the save file dialog</td>
</tr>
<tr>
<td>Settings</td>
<td>F7</td>
<td>Command - ,</td>
<td>Open the Settings dialog</td>
</tr>
<tr>
<td>Model Library</td>
<td>F6</td>
<td>F6</td>
<td>Open the Model Library Explorer dialog</td>
</tr>
<tr>
<td>Run</td>
<td>F9</td>
<td>F9</td>
<td>Run main file of current project with GAMS</td>
</tr>
<tr>
<td>Run with GDX Creation</td>
<td>F10</td>
<td>F10</td>
<td>Run main file of current project with GAMS and create a GDX file</td>
</tr>
<tr>
<td>Compile</td>
<td>Shift - F9</td>
<td>Shift - F9</td>
<td>Compile main file of current project with GAMS without execution</td>
</tr>
<tr>
<td>Compile with GDX Creation</td>
<td>Shift - F10</td>
<td>Shift - F10</td>
<td>Compile main file of current project with GAMS and create a GDX file without execution</td>
</tr>
<tr>
<td>Interrupt</td>
<td>F12</td>
<td>F12</td>
<td>(Gracefully) Interrupt currently running GAMS job</td>
</tr>
<tr>
<td>Stop</td>
<td>Shift - F12</td>
<td>Shift - F12</td>
<td>Stop currently running GAMS job</td>
</tr>
<tr>
<td>Show extended parameter</td>
<td>Shift - Ctrl - 3</td>
<td>Control - Option - 3</td>
<td>Open the extended GAMS parameter editor</td>
</tr>
<tr>
<td>Project explorer</td>
<td>Ctrl - 1</td>
<td>Command - 1</td>
<td>Open Project Explorer</td>
</tr>
<tr>
<td>Process Log</td>
<td>Ctrl - 5</td>
<td>Command - 5</td>
<td>Open Process Log</td>
</tr>
<tr>
<td>Help</td>
<td>F1</td>
<td>F1</td>
<td>Open the GAMS Help.</td>
</tr>
</tbody>
</table>

Actions and their shortcuts that can be performed via the left pane of the extended GAMS parameter editor are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert new parameter</td>
<td>Ctrl - Return</td>
<td>Ctrl - Return</td>
<td>Insert a new parameter</td>
</tr>
<tr>
<td>Delete selection</td>
<td>Ctrl - Delete</td>
<td></td>
<td>Delete the selected parameter</td>
</tr>
<tr>
<td>Delete all parameters</td>
<td>Alt - Delete</td>
<td></td>
<td>Delete all parameters</td>
</tr>
<tr>
<td>Move up</td>
<td>Ctrl - Up</td>
<td>Command - Up</td>
<td>Move the selected parameter up for 1 row</td>
</tr>
</tbody>
</table>
### Actions and their shortcuts that can be performed via the right pane of the extended GAMS parameter editor are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move down</td>
<td>Ctrl - Down</td>
<td>Command - Down</td>
<td>Move the selected parameter down for 1 row</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl - A</td>
<td>Command - A</td>
<td>Select all parameters</td>
</tr>
<tr>
<td>Show parameter definition</td>
<td>Ctrl - F1</td>
<td></td>
<td>Show definition of this selected parameter in the right pane</td>
</tr>
<tr>
<td>Show all parameters of the same definition</td>
<td>Shift-F1</td>
<td></td>
<td>Show all options of the same definition defined in the right pane</td>
</tr>
<tr>
<td>Resize columns to contents</td>
<td>Ctrl - R</td>
<td>Command - R</td>
<td>Resize the columns in the left pane to contents</td>
</tr>
</tbody>
</table>

### 6.13.3.4 Integrated Help

The Help View is designed to integrate the navigation of the GAMS documentation into GAMS Studio. Press F1 or choose Help > GAMS Documentation or check View > Help to start the Help View and browse the documentation. The start page of the Help View is the main document page available in the GAMS distribution that has been used to run GAMS Studio. See Home on how to always navigate back to the start page of the document. Alternatively, choose Help > Studio Documentation to directly jump to the start of the GAMS Studio documentation. Click Close button or uncheck View > Help to dismiss the Help View. GAMS Studio will remember the last viewed page along with its browsing history until GAMS Studio is restarted.

The Help View starts in docking state for the first time and can be docked around the editor in the central widgets area by either dragging the view to the desired location. Studio will remember last state of the Help View before it was closed.
Figure 6.34 Help View when docked to the right of Code Editor

The Help View can also be floated by double clicking the title part of the Help View. Dragging or double clicking the title of the Help View again will dock the Help View.

Figure 6.35 Help View when floated

Note that the Help View does not offer the full browsing features of a web browser. Though the help view
offers a Open this page in Default Web Browser button to use the full features of a web browser.

An overview of Help features:

- **Home**: Start the start help page [GAMSDir]/docs/index.html
- **Back**: Back to previous page
- **Forward**: Go to next page
- **Reload**: Reload the content of this page
- **Stop**: Stop loading the content of this page
- **Bookmarks**:
  - **Bookmark this page**: Store the title of the page as the name of bookmark together with its location. The entry of the bookmark will appear below the Organize Bookmarks section. Click on an entry to jump to the page that has been bookmarked.
  - **Organize Bookmarks**: Open the bookmark dialog to edit or delete a bookmark entry. Right click on the selected entry in the bookmark table, then either load the selected bookmark in the Help View, or delete the selected bookmark. The selected bookmark Name and Location can be edited from the lower part of the bookmark dialog, then press Enter. Press Esc to cancel the edit. Click close button to dismiss the bookmark dialog.
• **Zoom In**: Zoom in the page to increase the font size. Press Ctrl - +, or Ctrl - Mousewheel Up or choose View > Zoom In menu to zoom in the page.

• **Zoom Out**: Zoom out the page to reduce the font size. Press Ctrl - -, or Ctrl - Mousewheel Down or choose View > Zoom Out menu to zoom out the page.

• **Reset Zoom**: Reset the font size of the page to its original size. Press Ctrl - 0 or choose View > Reset Zoom menu to reset zoom in the page.

• **Help Option**:

  – **View This Page Online**: Browse the same documentation on GAMS website. This, for example, allows to use more sophisticated search functionalities across all GAMS documentation of version 40.1 or to browse different version of documentation, or to browse the GAMS website from the Help View.

  – **Open in Default Web Browser**: Browse the same document in default web browser. As the help view does not offer the full features of a web browser, this allows to browse the documentation using the full web browser features in default web browser.

  – **Copy page URL to Clipboard**: Copy of the current page's URL to the clipboard. This, for example, help to see the URL of the currently viewed page of the document.
The GAMS Call and Command Line Parameters

There are multiple ways to trigger the run of a GAMS model: Running a model via F9 from GAMS Studio, executing the GAMS Job. Run method in the object oriented APIs or calling the gams executable from a command line. In all cases the same GAMS engine runs the user’s model.

Several options are available to customize the GAMS run. Depending on the particular way GAMS is triggered these options are supplied in different ways. For example, in Studio via the parameter editor in the toolbar, in the object oriented API through the GAMS Options class, and from the command line via command line arguments. In order to avoid confusion with the often used word option we refer to these entities as command line parameters and demonstrate its use with the gams executable from the command line. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

In this chapter we will introduce how GAMS is called from the command line and how parameters may be specified on the command line or in other ways. In addition, we will discuss user-defined command line parameters, compile-time variables and compile-time constants, which are GAMS specialties, and environment variables in GAMS.

- **Find in page...**: Press Ctrl - F or choose Edit > Search menu to activate a search at the bottom area of the Help View. Type in a word to be found in the page. The word found in the page will be highlighted as it has been typed in and the number of found occurrences will be highlighted in the scrollbar of the Help View. Click previous button to find the previous occurrence, and click next button or press Enter to find the next occurrence. Check Case Sensitivity box to find a word case sensitively. Highlighting the word stays as the document is navigated from page to page until the keyword is clear or the search is dismissed or the Help View is closed or invisible. Click the close button or press Esc to dismiss the search.
Help on Specific widgets

It is possible to jump directly from directly from different widgets (Welcome Page, Code Editor, Listing Viewer, GDX Viewer, Reference File Viewer, and Solver Option Editor) to the related part of the document.

From Welcome Page, Listing Viewer, GDX Viewer, and Reference File Viewer, press F1 will activate the GAMS Studio page and jump to the section corresponding to the widget. From GAMS Parameter Editor, press F1 on a parameter entry in the extended parameter editor will activate the The GAMS Call and Command Line Parameters help page containing the detailed description of the GAMS parameter. From Solver Option Editor, press F1 on an option entry in the editor will activate the related solver page containing the detailed description of the option.

From Code Editor, it is possible to jump to the document page that describes Dollar Control Options or to the index page that lists entries related to Data Types and Definitions and Language Items in Code Editor, as well as from the parameter editor to the GAMS parameter described in The GAMS Call and Command Line Parameters:

- Press F1 on a Dollar Control Option within the editor will activate the page Dollar Control Options in the Help View and jump to the corresponding dollar control option described in Dollar Control Options chapter.
• Press F1 on a keyword in Data Types and Definitions or in Language Items within the editor to activate the Index Page in the Help View that lists the index entries related to the keyword.

• Press F1 on a parameter entry within the extended parameter editor to activate the help page displaying the detailed description of the GAMS parameters in The GAMS Call and Command Line Parameters.
Figure 6.36 Help View when press F1 on parameter entry 'output' in extended parameter editor

Figure 6.37 Help View when press F1 on parameter definition entry 'LimRow' in extended parameter editor

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom In</td>
<td>Ctrl- + or Ctrl - Wheel Up</td>
<td>Command - + or Command - Wheel Up</td>
<td>Zoom in the page to increase the font size</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>Ctrl- - or Ctrl - Wheel Down</td>
<td>Command - - or Command - Wheel Down</td>
<td>Zoom out the page to reduce the font size</td>
</tr>
<tr>
<td>Reset Zoom</td>
<td>Ctrl - 0</td>
<td>Command - 0</td>
<td>Reset the font size of the page to its original size</td>
</tr>
</tbody>
</table>
6.13.4.1 Summary of Actions and Shortcuts

6.13.4 MIRO

GAMS Studio can be used to develop models for GAMS MIRO. Please go to the GAMS MIRO documentation to learn more about the features and workflow of the MIRO integration.

6.13.4.1 GAMS MIRO Shortcuts

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run base mode</td>
<td>F8</td>
<td>F8</td>
<td>Run main file of current project with MIRO in Base Mode</td>
</tr>
<tr>
<td>Run configuration mode</td>
<td>Shift - F7</td>
<td>Shift - F7</td>
<td>Run MIRO configuration mode</td>
</tr>
</tbody>
</table>

6.13.5 NEOS

GAMS Studio can remotely execute models on the NEOS server. The model is compiled locally before the compilation object is send to NEOS. Note that a GDX file can be created but always has the name out.gdx. On running a dialog pops up to enable further setup:

- "Email" - The email to submit a job. It can be predefined by setting NEOS_EMAIL in the GAMS User Configuration or by defining the environment variable NEOS_EMAIL.
- "Create a GDX file" - Create a GDX file even if the Command line parameter is not set.
- "Short priority" - The execution may start earlier and provides intermediate log output, but it is limited to an execution time of five minutes.
- "Long priority" - The execution may start later and provides no intermediate log output, but it is not limited in execution time.

The selections of the dialog are stored to the settings. To reveal the Terms of Use after checked Don’t show Terms of Use again there is a checkbox in Settings > Misc.

The results of the remotely executed model are placed into a sub-folder named by the base name of the model. The log output of the remote execution has a different background color which can be configured in the settings dialog (Default Text, 3rd color). Links in the remote execution log output are replaced by their local counterpart.
6.13.6 GAMS Engine

GAMS Studio can remotely execute models on your own GAMS Engine server. The model is compiled locally before the compilation object is sent to GAMS Engine. Additional files can be sent to the server by creating a file with the name of the model and the extension ".efi" (for "External Files"). Each line in this file addresses an additional file. Files defined in the EFI file that should be actualized locally by changes from the GAMS Engine server can be marked by adding a < separated by a space.

Example: if an EFI file named trnsport.efi contains these lines:

inc/trnsport.inc
inc/data.txt <

and the model trnsport.gms is executed, both files will be transferred to GAMS Engine. After solving, the file inc/trnsport.inc will stay untouched but inc/data.txt will be replaced by the version returned from the server.

GAMS Studio provides an EFI editor to select the files. The context menu of a project in the Project Explorer allows to create or open the EFI file matching the current runnable GAMS model. With this editor files can be selected and selected files can be marked to be actualized locally. You can switch an open EFI editor to a text editor by selecting Reopen as text after changes have been saved.

On running a dialog pops up asking the user to log in. If already logged in before and the login is still valid this first page is skipped.

Login page

"Url" - The URL to your GAMS Engine server
"User" - The username to login
"Password" - The password to login

While changing the URL, Studio tries directly to access the server and requests version information. On success it gets the GAMS Engine version placed to the right and the version of the GAMS installation on the server to the left. If the local GAMS version is newer the command line parameter "previousWork=1" will be added automatically if no "previousWork" is already present. The entries of the dialog (excluding the password) are stored to the settings. Further configuration can be done in the settings dialog.

Being logged in, the submit page allows to setup the remote job:

Submit page

"Namespace" - The namespace to be used on the server
"Instance" - Allows to specify the user instance for GAMS Engine SaaS
"Create a GDX file" - Creates a GDX file even if the command line parameter is not set

![Submit page](image-url)
To get back to the login page click the "Logout" Button. The job is submitted to the GAMS Engine server on clicking "OK". To always run the same setup, the "OK, don't ask" can be selected. For this Studio session the job is always submitted directly without showing this dialog. To see the dialog again, use Ctrl-Click on the Engine icon, or click on "Settings > Remote > Reactivate Dialog".

The selection of an user instance and the information of available space and time quota is only visible on connecting to a GAMS Engine SaaS server.

The results of the remotely executed model are placed into the working directory:

- The LST and LXI files from the server are renamed to "<model>-server.lst" and "<model>-server.lxi" to protect the local files.
- All generated file will replace an existing file of the same name (like GDX, REF, and LOG files).
- All file entries from the EFI file that end with "<" will replace the local existing file.

The log output of the remote execution has a different background color which can be configured in the settings dialog (Default Text, 3rd color). Links in the remote execution log output are replaced by their local counterpart.

6.13.7 Dialogs and Actions

6.13.7.1 GAMS Licensing

The GAMS Licensing dialog is part of the GAMS Studio Help menu. It provides information about the used GAMS as well as available solvers, which lists the solver licenses and their capabilities. Furthermore, the dialog can be used to install a GAMS license. To do so the steps below have to be applied.

1. Copy the GAMS license to the clipboard
2. Open GAMS Studio and click on Help > GAMS Licensing
3. A messages box shows up if a GAMS license has been found on the clipboard. If 'Yes' is clicked the new license will be installed and presented via the GAMS Licensing dialog; otherwise the old license is shown.

Studio will also detect a license on the clipboard when it starts. If a license is detected the same message box will show up like if the GAMS Licensing dialog is used.

6.13.7.2 Check for Update

Check if a more recent version of GAMS or GAMS Studio is available for download (and can be used with the present license) and report the results. The information shown uses the current license file for this.

6.13.7.3 Search and Replace
<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open/Focus Search Dialog</td>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Opens the search dialog or focusses it if it is already open, if Text is selected it will automatically be entered as a search term</td>
</tr>
<tr>
<td>Find next</td>
<td>F3 or Enter</td>
<td>F3 or Enter</td>
<td>Jumps to the next match that fits all search criteria</td>
</tr>
<tr>
<td>Find previous</td>
<td>Shift - F3</td>
<td>Shift - F3</td>
<td>Jumps to the previous match</td>
</tr>
<tr>
<td>Find All</td>
<td>Shift - Enter</td>
<td>Shift - Enter</td>
<td>Starts a search, opens the page of search results and jumps to the first hit</td>
</tr>
<tr>
<td>Close Dialog</td>
<td>Esc</td>
<td>Esc</td>
<td>Closes the search dialog. On second press any highlights of results will be removed</td>
</tr>
</tbody>
</table>

Pressing the Ctrl - F opens the Search and Replace window. If text is selected it will be pasted into the search field. Users can search specific files with it and do text replacement operations. Depending on which file type is opened in the editor, the search widget changes to visualize which actions are available. Due to technical limitations .gdx files cannot be searched. Also, replacing in read-only file types (e.g. .lst or ref) is deactivated. For reasons of performance the search stops when reaching 50000 search results. This is visualized in all places where the number of matches is shown by showing ”50000+” as the results number, indicating that there might be more results.

In the first row of the dialog there is the Search field, which takes the search term and saves a list of recent searches. Next to it is the ”Clear” button that removes all highlights, results and clears both text fields. A shortcut to clear all results is to press the Esc key in the Code Editor. The next row houses items related to the replace functionality. Most of these are deactivated for files that are opened in read-only mode (e.g. .lst). First comes the replace input field where users can input the text which replaces the search term. The next button is ”Replace”: On the very first click it selects the next match, without replacing anything. On the next click it replaces the current selection with the replacement term and jumps to the next match in the same file, selecting it. Users can use this to click through a file, replacing words while keeping an overview over what is actually changed. Replacement actions can be undone by pressing either the undo button or hotkey Ctrl - Z.

"Replace All" replaces all matches in the currently selected scope after a pop-up is shown asking the user whether the replacement of n occurrences of X with Y is intended. In this pop-up the user can also decide...
to start a search instead, opening the results page showing all occurrences of the would-be replace action. Then come the options to narrow down a search. "Whole Words" excludes partial hits from matches. For example, when searching for "in" with this option checked only the word "in" is found but no occurrences in "information". "Case Sensitivity" sets if the case of letters in the search term matters. Reminder: The GAMS language is case insensitive, so this option can actually lead to incomplete results, depending on what it is used for. "Use Regex" activates an advanced search term interpretation mode and stands for **Regular Expressions**. When activated, instead of a single search term users can specify a pattern that matches an array of different words. Click here for further information about regular expressions.

In the next row is the scope selector. The scope of the search or replace action can be set here with five options available:

- "Selection" searches the text selected in the current file. The search dialog will give feedback if no selection is present. To remove the search selection press either Esc or the "Clear Selection" button. Keep in mind that starting a new search moves the search selection to the currently selected text.

- "This File" limits the search to the currently active file and is the default.

- "This Project" searches all files that belong to the same project as the currently opened file. These are all children of the same project in the Project Explorer.

- "Open Tabs" searches all files currently opened in Studio (except GDX files).

- "All Files" searches all files that appear in the Project Explorer and are searchable.

- "Folder" searches all files in a specified folder that can be located anywhere on disk and does not need to be opened in Studio beforehand. When opening a result the file will be added to the Project Explorer.

Next to it are the find previous occurrence and find next occurrence buttons, labeled "<" and ">") respectively. These will find and highlight the next word matching the search criteria either before or after the current text cursor position. Next to it there is the "Find All" button which starts a search in the scope selected by the user. In the output pane of studio a table containing all matches will open. Items can be double clicked to perform a jump to the result and the up and down arrow keys can be used to select quickly select and jump to results. Information about the file, the location of the match, plus some context information are also shown.

Depending on which scope is selected, additional elements of the search dialog will be shown:
The first section is the file filter section which is available for all modes except "Selection" and "This File". It gives the user the possibility to limit the search to certain files (Include Filter) but exclude others (Exclude Filter). Filenames can be specified by using a wildcard syntax. See the tooltips for some simple examples. The drop-down menus come with a few default options for GAMS specific file types but can also be used to enter own patterns. Comma-separated lists of patterns are also supported. The second section is only available when "Folder" scope is selected and lets users set the path to search. It is a drop-down style textfield, so like the others it keeps a history of recent entries. Next to it is a browse button that opens the operating systems default folder picking dialog to easily find a path. Tooltips are available for most items in the search window and contain further information like keyboard shortcuts.

### 6.13.7.3.1 Workflow Tips

- Edit a file, hit Ctrl - F to open the search widget. The search field is focused automatically so you can start typing your search term. Pressing the Enter key is a shortcut for Find Next. Press Enter again or F3 to step through all matches. Press Shift - F3 to step backwards. Make changes to you document, press Ctrl - F again to re-focus the search widget without having to pick up the mouse.

- Enter a search term, and then Shift - Enter to trigger “Find All”. This will open a list of results.

- The Esc key will close the search widget. Pressing Esc again will clear all highlighted search results.

- Use project or file prefixes when working with projects with many files. Use search filters to only search relevant files.
6.13.7.4 Model Library Explorer

The Model Library Explorer is used to search the different model libraries provided by GAMS and to retrieve their models in a convenient way. It can be opened either by choosing GAMS > Model Library Explorer from the menu, by using the shortcut F6 or by clicking the corresponding icon in the Quick Access Toolbar.

Every library is presented in a separate tab. The search facility in the upper part of the Model Library Explorer allows for dynamic filtering of all model libraries simultaneously. As the search string is entered, the results are applied to the tabs representing the different libraries. If there are no more results on the selected model library tab, the first non-empty tab is automatically selected. This is undone as soon as the originally selected model library contains results again. The parenthesis enclosed numbers indicate the number of models found for the specific library and the current search input. More about the filter can be found in the Filter Section.

![Figure 6.40 Searching for trns in the Model Library Explorer](image_url)

Beside a short description, several models have a longer and more detailed description available. Selecting a model and clicking on the Description button in the lower right corner opens a pop-up dialog showing further information about the model. A model can be opened either by clicking on the Load button, by double clicking or by selecting the model and pressing the Enter key.

6.13.7.4.1 User Libraries  Additionally to the model libraries distributed by GAMS, it is possible to access user defined model libraries by providing a GLB file along with the corresponding files belonging to the models in the library. The Model Library Explorer looks into a specific location for user defined model libraries. This location can be accessed or changed by opening the Settings dialog (File > Settings) and switching to the Misc. tab. The Arrow button will show the location where the Model Library Explorer tries to find user defined model libraries, the Folder button allows to change the location. Adding a new library is done by copying the required files into a subdirectory in this location. See Creating a User Library on how to create a custom model library. Newly added libraries require a restart of the Model Library Explorer to become visible.

6.13.7.5 Settings

The GAMS Studio settings dialog can be accessed via the File > Settings menu entry or with the hotkey F7. Users can change certain aspects of Studio like behavior or appearance. Settings are categorized loosely on five tab pages.
6.13.7.5.1 General page  On the ”General” tab users control general settings of Studio. Some additional hints:

- ”Default GAMS Studio workspace”. This path is used as a default location for models imported from the Model Library Explorer.
- ”Automatically save modified files before running GAMS” is an option that saves all modified files when the user starts a GAMS run. Thereby, previously unsaved changes will be incorporated into that GAMS execution without the need to go through all files and make sure that changes were saved manually.
- ”Open .lst file after running GAMS” automatically puts the generated lst file into the foreground after each GAMS run.
- ”Jump to first compilation error” is an automatism that aims to make working with Studio a little easier. When compilation errors occur, the editor and the log both jump to the first error detected if this option is activated. This also applies when the option Open .lst file after running GAMS is activated. Studio then opens the lst file generated and jumps to the first error in it. If you navigate back to the gms file, the view has also moved to the first error found.
- ”Open file in current project by default” switches the menu File > Open (Ctrl-0) to always open a file in the current project, even if it exists in another project. The alternative open (Ctrl-Shift-0) is changed to File > Open in new project....
- ”Open file finds or creates new project” switches the menu File > Open (Ctrl-0) to find the file in any project and open it. If not found, a new project is created. The alternative open (Ctrl-Shift-0) is changed to File > Open in current project....
- At the bottom of this tab there is a section to backup all settings. When pressing the Export button a dialog pops up, asking for a filename. This name can be chosen freely and the file can than be copied to a USB stick for sharing, or just be saved for later. This is handy if you are planning a fresh install or want to copy your personal settings to a different machine. The button Import is used to load a saved settings file. These settings are applied immediately and the settings dialog will be closed automatically.

6.13.7.5.2 Editor & Log page  The ”Editor & Log” tab contains many self explanatory appearance options. Most notably font settings which is used for all monospace fonts like text editors and logs. The font size is used also to calculate the size of proportional fonts (see Zoom Groups)

- ”Enable auto-indentation” activates smart line break behavior. When pressing the Enter key a line break is inserted and the indentation of the line of origin is copied instead of moving the cursor to the first column.
- ”Highlight word under cursor without selection” changes the highlighting behavior of word occurrences. While the default behavior highlights other occurrences of the same word when double clicking or selecting a word, activating this option changes the behavior so that every time there is a word under the text cursor, all of its occurrences will be highlighted. Only full words and no partial matches are highlighted.
- ”Tab stop size” lets the user decide how many tabs will be inserted when pressing the Tab-key. GAMS Studio always replaces inserted tabs with spaces.
- ”Fold DCO blocks on open” folds all foldable blocks of the types ”Text”, ”Echo”, ”Put”, and ”Fold”
- ”Clear process log before GAMS execution” empties the log before running a gms file. If deactivated old log output is kept but colored in a lighter gray so users are still able to identify that they are looking at an old run instead of the most current one. One can also decide if the log should be written to disk as well and if so, how many old versions of the same log should be kept on disk as backup. Note, that in addition to this log file written by Studio, one can instruct GAMS itself to write a log file using the GAMS parameter logOption, which could cause a file naming clash.
6.13.7.5.3 GDX Viewer

- "Default Symbol View" controls if the list view or the table view is used as the default view when a symbol is displayed for the first time.

6.13.7.5.4 Colors page The "Colors" tab enables you to configure several colors. Studio provides two fixed themes: the "dark" theme and the "light" theme. Depending on the your operating system, this can be selected by the system and/or manually. You can use this to set Studio to use a darker color palette for when you like to work in the dark and don't like to get blinded by bright windows.

![Image of Studio using a dark theme](image.png)

**Figure 6.41 Studio using a dark theme**

The fixed themes can't be modified, modifying a value implicitly creates a copy of the theme where the changed value is applied. After copying a theme, the new user theme can be adapted to your needs. Additionally these user themes can be exported and imported to a file each.
The colors are listed in two sections:

- On the "Syntax colors" page you can define a text color and the attributes bold and italic.
- The "Editor colors" page allows to change the colors of text, background, and second background (for blinking). The very first entry on this page, the "Default Text", is the base of the syntax colors.

6.13.7.5.5 Remote page  The "Remote" tab is used to set various remote settings.

- In the MIRO section the location of your local MIRO installation can be defined.
- The checkbox "Always agree to NEOS" allows to hide the "Terms Of Use" in the submission dialog of NEOS.
- In the GAMS Engine section it can be defined if a login token should be stored to the settings to stay logged in on restarting Studio. The next line allows to define how long the login is valid.
- If the dialog has been switched invisible by clicking "OK, don't ask", the button "Reactivate dialog" switches it on again.
6.13 GAMS Studio

6.13.7.5.6 Misc page  The "Misc" tab contains some lesser common settings.

- "GAMS source file extensions" allows to add own file extensions that Studio counts as runnable GAMS. File types added here can be executed, set as main file, and are assigned to the GAMS Syntax Highlighter. The field accepts a comma separated list of file extensions.

- "Auto-reload extensions" Specific extensions that should be reloaded automatically can be added here. Files of that type are reloaded into the editor if changed externally. If they also were changed internally, a dialog asks how to proceed. The field expects a comma separated list of file extensions.

- "User model library" shows the folder that is used to find user generated model libraries, see User Libraries. This folder can be changed to a custom location. Up to 10 locations are stored in the history. To remove the current entry from the history use the trash-bin button. The arrow-button opens the file browser in the current folder.

- "Reset History" clears the list of last files used on the Welcome Page

- "Reset Window" resets the appearance of Studio if something went wrong.

- The next group contains a list of options for the Solver Option Editor.

6.13.7.6 Delete scratch directories

GAMS creates working folders for running processes in your working directory. Normally these are automatically removed when a run finishes. However, if a run was interrupted sometimes these working directories remain. That in itself is not a problem as GAMS will pick a new name for the next run if the default name for the working directory is already taken. To automatically clean up the user workspace, Studio offers an action to do this. In the menu under GAMS > Delete scratch directories the user has the possibility to remove unwanted, autogenerated directories. Before deleting anything, there will be a popup asking the user for confirmation and also showing the path of the directory that will be cleaned. If the user choses "Yes", all directories that start with 225 followed by one or two letters will be deleted in the given path, which is usually the default workspace. This action is automatically triggered when GAMS has exhausted the pool of possible scratch directory names, but not without asking the user for confirmation first. To avoid any potential errors this action should only be triggered if there are no running GAMS jobs at the moment.

6.13.7.7 Full Screen & Distraction Free Mode

Especially small devices suffer from sparse screen real estate. Studio comes with a few features to make the most of the space that is available on your machine.

The Full Screen mode is activated by pressing Alt-Enter on Windows and Linux systems, while on macOS the shortcut is Command-Control-F. This enlarges Studio over the whole screen, covering the taskbar and other operating system specific items. The same shortcuts also exit Full Screen mode.

As an alternative, there is also the Distraction Free mode. Some users want to keep their taskbar so with the shortcuts Ctrl-Alt-Enter on Windows and Linux orCmd-Option-Enter on macOS hide all Studio widgets and enlarge the central widget to the size of the studio window. You can use this to get the log-, file- and help view out of the way and concentrate writing. Pressing the shortcut again brings everything back. Running your GAMS modell in the Distraction Free mode will also open the log, so you are not missing out on anything.

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter/Exit Full Screen</td>
<td>Alt - Enter</td>
<td>Command - Control - F</td>
</tr>
<tr>
<td>Enter/Exit Distraction Free Mode</td>
<td>Ctrl - Alt - Enter</td>
<td>Command - Option - Enter</td>
</tr>
</tbody>
</table>
6.13.7.8 GDX Diff

The GDX Diff dialog serves as a graphical interface to the command line tool GDXDIFF. It compares the data of two GDX files and writes the differences to a third GDX file. The dialog can be opened by choosing Tools > GDX Diff from the menu.

![GDX Diff Dialog](image)

The dialog consists of two parts. The upper part is used to specify the two GDX files for the comparison. The resulting GDX difference file is determined automatically as long as no manual changes have been applied. The default schema for the resulting GDX difference file is the location of the first input GDX file and the file name `diff.gdx`. The difference file will be updated whenever the first input is modified to an existing file. If the GDX Diff dialog gets opened and does not contain a valid first input file, the most recently used directory is used as the location for the resulting difference file instead.

The lower part of the dialog contains different options that can be used in order to control the behavior of the comparison:

- **Eps**: Absolute difference for comparisons. If the difference between two values exceeds Eps, a difference will be reported. Default is 0.0.
- **RelEps**: Relative difference for comparisons. If the value of RelEps is exceeded, a difference will be reported. Default is 0.0.
- **Field To Compare**: The specified subfield is the only field used for deciding if a variable or equation is different. List of possible values:
  - **All**: All fields
  - **L**: Level
  - **M**: Marginal
  - **Lo**: Lower Bound
  - **Up**: Upper Bound
  - **Prior
  - **Scale**
6.13 GAMS Studio

- **Ignore Set Text**: Allows to ignore explanatory text of set elements.

- **Diff Only**: Differences for variables and equations will be written as parameters. Each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option can not be used in combination with Field Only.

- **Field Only**: Used in combination with Field To Compare. The variables and equations will be written as parameters for the selected subfield. This option can not be used in combination with Diff Only and it requires a value for Field To Compare other than All.

As soon as the OK button is clicked, a validation check on the given input is performed and the two input files are compared. If the specified difference file is already open in a GDX Viewer, the corresponding tab remains open while the underlying GDX file is detached. As soon as the file was updated, the GDX Viewer is reinitialized. In case the GDX file was not open already it gets opened in a new tab. The Cancel button closes the dialog and stops a process that has not been finished yet. The dialog needs to be kept open for longer running comparisons of large input files. The Reset button can be used to reset the whole dialog to its default state which also enables the automatic pre-population of the difference file path in case it has been disabled before due to a manual change.

The GDX Diff dialog also has an integration in the context menu of the Project Explorer. As soon as either exactly one or two GDX files are selected, the context menu contains an entry for opening the GDX Diff dialog with the file(s). If only one file is selected, the input field that is empty is populated with the corresponding file path. If both fields are filled, the first input file path gets overwritten. In case of two selected GDX files, both input fields get overwritten.

### 6.13.8 Terminal

GAMS Studio is able to open a native terminal via the main menu Tools > Terminal or the shortcut CTRL - T. Alternatively, a terminal can be opened by right clicking on any entry of the "Project Explorer" and selecting Open terminal. The terminal points to the workspace of the current project. On macOS the terminal location is always the user's home directory.

### 6.13.9 Command Line Options

GAMS Studio can be started with additional command line options that change the behavior or trigger certain functionality at start up time. The syntax for starting GAMS Studio from the command line is

```
studio [options] [files]
```

While files is one or more files to be opened by GAMS Studio, the following options can be supplied:

- `-?, -h, --help`: Displays the help.
- `-v, --version`: Displays version information.
- `--ignore-settings`: Ignore settings files. Studio will load with default settings without writing them.
- `--reset-settings`: Reset all settings to default. Studio will load with default settings and save them.
- `--reset-view`: Reset studio visually without resetting settings. Useful when a widget got lost or studio started on a disconnected monitor.
- `--gams-dir path_to_gams`: Point Studio to a specific GAMS installation.
6.13.10 General Shortcuts

This section lists any shortcuts that are global to Studio. Shortcuts for more specific areas are mentioned in their corresponding section.

6.13.10.1 Studio Workflow Shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl - 1</td>
<td>Command - 1</td>
<td>Set focus to the project explorer. After that the arrow keys can be used for file navigation.</td>
</tr>
<tr>
<td>Ctrl - 2</td>
<td>Command - 2</td>
<td>Set focus to main editor.</td>
</tr>
<tr>
<td>Ctrl - 3</td>
<td>Command - 3</td>
<td>Set focus to the command line option input field.</td>
</tr>
<tr>
<td>Ctrl - 4</td>
<td>Command - 4</td>
<td>Opens the tab browser which has a list of all open tabs. Focus is on an input field that can be used to filter tabs. Pressing Return opens the highlighted file.</td>
</tr>
<tr>
<td>Ctrl - 5</td>
<td>Command - 5</td>
<td>Set focus to log widget.</td>
</tr>
<tr>
<td>Shift - Ctrl - 3</td>
<td>Control - Option - 3</td>
<td>Opens and closes extended command line option editor.</td>
</tr>
<tr>
<td>Ctrl - Tab</td>
<td></td>
<td>Toggles through the open tabs of the central widget from left to right.</td>
</tr>
<tr>
<td>Ctrl - Shift - TAB</td>
<td></td>
<td>Toggles through the open tabs of the central widget from right to left.</td>
</tr>
<tr>
<td>Alt - Return</td>
<td>Command - Control - F</td>
<td>Full Screen</td>
</tr>
<tr>
<td>Ctrl - Shift - T</td>
<td>Command - Shift - T</td>
<td>Restore recently closed tab</td>
</tr>
<tr>
<td>Ctrl - Shift - W</td>
<td>Command - Shift - W</td>
<td>Close all tabs</td>
</tr>
<tr>
<td>Ctrl - W</td>
<td>Command - W</td>
<td>Close current tab. Alternatively, a tab can be closed by clicking the middle mouse button.</td>
</tr>
<tr>
<td>Ctrl - Q</td>
<td>Command - Q</td>
<td>Close GAMS Studio</td>
</tr>
</tbody>
</table>

6.13.10.2 Studio Dialogs

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F1</td>
<td>Open the GAMS Help. If the word under the cursor is a GAMS keyword a search in the documentation document will be started</td>
</tr>
<tr>
<td>F6</td>
<td>F6</td>
<td>Open GAMS Model Library Explorer</td>
</tr>
<tr>
<td>F7</td>
<td>Command - ,</td>
<td>Open Studio Settings</td>
</tr>
<tr>
<td>Ctrl - F</td>
<td>Command - F</td>
<td>Open search and replace widget</td>
</tr>
<tr>
<td>Ctrl - G</td>
<td>Command - G</td>
<td>Goto specific line number</td>
</tr>
<tr>
<td>Ctrl - N</td>
<td>Command - N</td>
<td>Open new file dialog</td>
</tr>
<tr>
<td>Ctrl - O</td>
<td>Command - O</td>
<td>Open file open dialog</td>
</tr>
<tr>
<td>Ctrl - Shift - O</td>
<td>Command - Shift - O</td>
<td>Open file dialog to open the file(s) in the current (a new) project</td>
</tr>
<tr>
<td>Ctrl - P</td>
<td>Command - P</td>
<td>Open print dialog</td>
</tr>
<tr>
<td>Ctrl - S</td>
<td>Command - S</td>
<td>Save the current file. If the file does not exist on disk open the save file dialog</td>
</tr>
<tr>
<td>Ctrl - Alt - S</td>
<td>Command - Option - S</td>
<td>Open save as dialog</td>
</tr>
<tr>
<td>Ctrl - Shift - S</td>
<td>Command - Shift - S</td>
<td>Save all open files. Open save file dialog if needed</td>
</tr>
</tbody>
</table>
### 6.13.10.3 Studio Recovery

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl - F2</td>
<td>Command - F2</td>
<td>Reset Studio to default visual settings (window position and -size, widget position and -visibility) Useful when a widget got lost or studio started on a disconnected monitor.</td>
</tr>
</tbody>
</table>

### 6.13.10.4 Zoom Editor and View

All editors and views can be zoomed in and out. To avoid repeating action a zoom is applied to all editors and views of the same group. Beneath the [Integrated Help](#) and the [Welcome Page](#) these groups are defined:

- **Text**: contains all editors and views that show flowing text in monospace fonts and appear as [Central Widget](#), like the [Code Editor](#) and the [Listing Viewer](#).
- **Log**: contains the [Process Logs](#) and the [System Log](#).
- **Table**: contains all editors and viewers that show table and tree elements in proportional fonts as a [Central Widget](#), like the [GDX Viewer](#) and the [Solver Option Editor](#). It additionally contains the [Command Line Option Editor](#).

<table>
<thead>
<tr>
<th>Action</th>
<th>Shortcut</th>
<th>macOS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom In</td>
<td>Ctrl - + or Ctrl - = or Ctrl - Wheel Up</td>
<td>Command - + or Command - = or Command - Wheel Up</td>
<td>Zoom in all editors and views of the same group</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>Ctrl - - or Ctrl - Wheel Down</td>
<td>Command - - or Command - Wheel Down</td>
<td>Zoom out all editors and views of the same group</td>
</tr>
<tr>
<td>Reset Zoom</td>
<td>Ctrl - 0</td>
<td>Command - 0</td>
<td>Reset zoom of all editors and views</td>
</tr>
</tbody>
</table>

### 6.13.11 Usage Hints

#### 6.13.11.1 Filter in Tables and Trees

To find relevant entries in tables and tree views Studio provides a special line edit containing up to four buttons aligned to the right:

- **Clear** - clears the term in the line edit.
- **Exact Match** - when active, the filter accepts only rows with one cell matching exactly the term regarding wildcard or regular expression setting
- **Regular Expression** - when inactive, the wildcards ’*’ (any amount of characters) and ’?’ (one character) can be used. Otherwise the filter provides full regular expression syntax. [Click here](#) for further information about regular expressions.
- **All Columns** - when inactive, the filter checks only the key column. When active, all cells of the table/tree-view are taken into account.

![Figure 6.44 Filter LineEdit inactive and active](image)

### 6.13.12 System Requirements

Compared to most of the GAMS system, GAMS Studio has some additional system requirements, which depend on the platform used. Also, additional information about technical requirements are listed at the [GAMS Studio GitHub Wiki](https://github.com/gams-studio).

#### 6.13.12.1 Linux

The Linux version of GAMS Studio is distributed as **AppImage** and requires glibc 2.27 as well as **FUSE** to be installed.

The GAMS Studio AppImage was tested and it was verified that it worked for the following Linux distributions (which are all under maintenance). All Linux distributions are under constant development and updates may change the behavior of the AppImage. Please check the [Studio issue page](https://github.com/gams-studio) if you encounter any issue.

- Arch
- CentOS Stream 8
- Debian 10
- Debian 11
- Fedora 36
- openSUSE Leap 15.3
- openSUSE Tumbleweed
- Ubuntu 18.04 LTS
- Ubuntu 20.04 LTS
- Ubuntu 22.04 LTS

Other flavors might work as well, but were not tested. If you experience problems with a particular Linux distribution, please let us know.

In some cases the Studio AppImage might not run. If this is the case extract the AppImage by running

```
$ ./studio.AppImage --appimage-extract
```

via the command line and start Studio from the extracted directory.
6.13.12.2 macOS

macOS has some special platform requirements related to finding GAMS. Below the steps are listed which are used to link GAMS Studio to GAMS.

- GAMS Studio is going to find GAMS in /Applications/** and /Users/username/Applications/** if it is part of a GAMS folder. It is required that GAMS Studio gets copied via Finder to its target location. This is the case if the GAMS installer is used or if the GAMS Studio DMG is open and GAMS Studio gets copied from there.
- If GAMS Studio could not find GAMS like described previously then it will search /Applications/** for a valid GAMS installation, i.e. a GAMS installation which has the minimum required GAMS version or higher.
- GAMS Studio will check the $PATH to find GAMS. In almost all cases this will not work on macOS.
- In case of special needs the path to GAMS can be set explicitly by starting GAMS Studio with the command line argument --gams-dir.

6.13.12.3 Windows

There are dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio, please run the appropriate installer for these libraries called vcredist_x64.exe, which can be found in the GAMS installation folder at [GAMS system]\studio.

6.13.13 Comparing GAMS Studio and GAMSIDE

As mentioned above, the classic GAMS IDE has been shipped with the GAMS system for many years and is still the workhorse for many GAMS programmers. So it is still around and an alternative to GAMS Studio, especially since it has features, which are not available in GAMS Studio (yet), but it also lacks some features which are available in GAMS Studio. The following table gives a compact overview about most of the differences.

<table>
<thead>
<tr>
<th>GAMSIDE only</th>
<th>GAMS Studio only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Platforms</td>
<td></td>
</tr>
<tr>
<td>- Windows</td>
<td>- Windows</td>
</tr>
<tr>
<td>- Linux</td>
<td>- Mac OS</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GDX Viewer</td>
<td></td>
</tr>
<tr>
<td>- Creating charts</td>
<td>- Filtering (both, symbols and data, labels and values)</td>
</tr>
<tr>
<td>- Writing to Html/Excel</td>
<td>- Sorting by numerical value</td>
</tr>
<tr>
<td>- Heat-map functionality (aka Data Colors)</td>
<td>- Option to change display format of numerical values</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Editor</td>
<td></td>
</tr>
<tr>
<td>- Fixed encoding (ISO 8859-1)</td>
<td>- Allow change of encodings and use UTF-8 by default</td>
</tr>
<tr>
<td>- Spell checker</td>
<td>- Block edit</td>
</tr>
<tr>
<td>- Recognize URLs and open them in the browser</td>
<td>- Help integration (pressing F1 on a keyword opens the relevant index page of the documentation)</td>
</tr>
<tr>
<td>- Different set of keyboard shortcuts</td>
<td>- Tooltips for compilation errors</td>
</tr>
<tr>
<td>GAMSIDE only</td>
<td>GAMS Studio only</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>- Navigation history</td>
<td>- Code Completion</td>
</tr>
<tr>
<td>- &quot;Smart typing&quot;: Automatic insertion of closing</td>
<td>- &quot;Smart typing&quot;: Automatic insertion of closing</td>
</tr>
<tr>
<td>character for brackets and quotes</td>
<td>character for brackets and quotes</td>
</tr>
<tr>
<td>- Distraction Free Mode</td>
<td>- Distraction Free Mode</td>
</tr>
<tr>
<td>- Code Folding</td>
<td>- Code Folding</td>
</tr>
<tr>
<td>- Open $[bat/lib]include files directly from</td>
<td>- Open $[bat/lib]include files directly from editor</td>
</tr>
<tr>
<td>editor</td>
<td></td>
</tr>
</tbody>
</table>

### Execution

- IDE Projects
- Studio Projects (see below for details)

- Set default options for the IDE only
- Interactive editor for the GAMS User Configuration

- Use different GAMS versions
- Interactive editor of GAMS command line parameters

- Use alternative license
- Option to jump to first error automatically

- Execute program based on extension
- Option to run with GDX creation automatically (F10 vs. F9)

- Solver selection menus
- Integration of GAMS MIRO

- Remote execution on NEOS Server for Optimization
- Remote execution with GAMS Engine

- Support of standard locations, e.g., to detect the license file at a standard path
- Option to not write log to disk

### Other

- Text Diff
- Welcome page

- Script recording
- Project Explorer

- Email file
- Reference File Viewer allows to jump to all references etc. and not just the first one

- Save test files in UNIX format
- Filtering in Model Library Explorer (all model libraries at the same time)

- Link to McCarl chm
- Allow to switch between light, dark or custom themes

- 32-bit application
- 64-bit application

### 6.13.13.1 GAMSIDE Projects vs. GAMS Studio Projects

With the GAMSIDE one always started with creating a "Project". This mainly defined a directory used to execute the models added to it (so, it is used to implicitly define the workDir when starting GAMS). When starting the IDE for the first time, one is asked to create a new project. For example, one creates this in directory `a`. After adding models `a/a.gms` and `b/b.gms` both get executed in the project directory `a`, meaning that data sources are expected in `a` (or relative to it) and also output files are generated in `a`, and not necessarily at the location of the `gms` file.

When starting GAMS Studio for the first time, it starts with an empty Project Explorer. Opening a new file creates a new Project per default. Studio Projects also have a "working directory". Initially, this is set automatically to the location of the file that was opened to create the new project. If a model gets executed, it gets executed in this working directory. So, if ones opens `a/a.gms` and `b/b.gms`, by default, Studio will create two projects: `a` and `b`. When `a` is started input files are expected in directory `a` and output files get generated in `a`. When `b` is started input files are expected in directory `b` and output files get generated in `b`. If one has structured the models so that they rely on the "GAMSIDE way" to execute
a model, one can just change the working directory in the project options. Also, it is possible to start with an empty project and just define the working directory explicitly in the way the GAMSIDE did it using the entry New Project from the File menu.

It is also possible to have multiple files in one project. For example, one could add b.gms to project a by drag’n’drop in the Project Explorer. Now, the concept of the ”main file” of projects becomes important: While the GAMSIDE always executed the currently active .gms file, GAMS Studio always executes the main file of the currently active project. So, in our example, you can look at a.lst and press F9 to rerun a.gms (the GAMSIDE would not run anything in this case). After adding b.gms to project a, one would also execute a.gms by pressing F9 while looking at b.gms. If you want to execute b.gms instead, you can make it the main file of the project. For this, right-click in the Project Explorer at b.gms and select ”Set as main file”. Note that the green arrow indicating the main file of a project in the Project Explorer switches from a.gms to b.gms. As a reminder: Projects do not know a particular working directory. Pressing F9 after switching the main file will execute b.gms in folder b.

![Figure 6.45 Switching the main file](image)

### 6.14 GDX2ACCESS

#### 6.14.1 Overview

GDX2ACCESS is a tool to dump the contents of a GDX file to an MS Access file (MDB or ACCDB format). Every identifier gets its own table in the database.
6.14.2 Usage

gdx2access inputFile {options}

The .gdx file extension of the inputFile can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.14.3 Options

Options are specified in an INI file and not directly on the command line. By default, the file gdx2access.ini located in the same directory as gdx2access.exe is consulted. If this file is not available, the program will continue using default settings (listed in the tables below).

It is also possible to tell the program to use a different INI file. This is done by using an extra argument of the form @iniFile. If you want to dump the contents of myFile.gdx to an MS Access file according to the options specified in a file called myIniFile.ini, run the following code:

```
gdx2access myFile.gdx @myIniFile.ini
```

In this case, the program will not read gdx2access.ini located in the same directory as gdx2access.exe, but rather myIniFile.ini in the current directory.

The INI file can contain two sections: [settings] and [debug]. A complete INI file with all possible settings looks like:

```
[settings]
scrdir=c:\tmp
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=0.0
undf=0.0
scalarTable=1
etFlag=1
dbVersion=9
```

Note that the values above are not the default values for some options!

Section [settings]

Below some short descriptions for the options in the [settings] section:
<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrdir</td>
<td><code>&lt;winodwstemp&gt;</code></td>
<td>Directory for temporary scratch files.</td>
</tr>
<tr>
<td>inf</td>
<td><code>1.0e100</code></td>
<td>The value used for GAMS special value <code>+INF</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>mininf</td>
<td><code>-1.0e100</code></td>
<td>The value used for GAMS special value <code>+INF</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>eps</td>
<td><code>0.0</code></td>
<td>The value used for GAMS special value <code>EPS</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>na</td>
<td><code>0.0</code></td>
<td>The value used for GAMS special value <code>NA</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>undf</td>
<td><code>0.0</code></td>
<td>The value used for GAMS special value <code>UNDEF</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>scalarTable</td>
<td><code>0</code></td>
<td>Possible values: 0 or 1. When set to 1, scalars of the same type are combined in a single table. The names for these tables are fixed: <code>ScalarParameter</code>, <code>ScalarEquation</code> and <code>ScalarVariable</code>. See also the example <code>Special value mapping</code>.</td>
</tr>
<tr>
<td>etFlag</td>
<td><code>0</code></td>
<td>Possible values: 0 or 1. When set to 1, include the text strings for sets containing element texts. See also the example <code>Writing Explanatory Text to Database</code>.</td>
</tr>
<tr>
<td>dbVersion</td>
<td><code>0</code></td>
<td>Specify the format of the output database. See also the example <code>Writing Explanatory Text to Database</code>. 0: Create database using current default format. 9: Create a database in the Microsoft Access 2000 (.mdb) file format. 10: Create a database in the Microsoft Access 2002-2003 (.mdb) file format. 12: Create a database in the Microsoft Office Access 2007 (.accdb) file format.</td>
</tr>
</tbody>
</table>

**Section [debug]**

Below some short descriptions for the options in the [debug] section:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td><code>5</code></td>
<td>Select an algorithm to insert data into the Access database. 1: Write a CSV file and use the TransferText action to read this into Access. This is fast, but does not always work when non-US language settings are used. 2: Use recordset.add to add records. This is slow, but does not use intermediate files. 3: Write a tab delimited file with a complete file specification (schema.ini) and use the ISAM Text driver to import the data. This is fast and should work in international settings. 4: For small data use method=2 and for larger data items use method=1. 5: For small data use method=2 and for larger data items use method=3.</td>
</tr>
<tr>
<td>thresholdCount</td>
<td><code>5</code></td>
<td>When to change between algorithms while using method=4 or method=5. The default is 5 records.</td>
</tr>
<tr>
<td>keepFiles</td>
<td><code>0</code></td>
<td>Possible values: 0 or 1. When set to 1, the program will not delete intermediate scratch files.</td>
</tr>
</tbody>
</table>
6.14.4 Examples

6.14.4.1 Intro

Suppose we want to write the data of the `trnspor` model from the GAMS model library after solving to an MS Access data file. First of all, we solve the model by running the following command. Note that we will save the complete symbol table to a GDX file called `trnspor.gdx` by adding the command line option `gdx=trnspor`.

```
gams trnspor gdx=trnspor lo=2
```

The option `lo=2` causes GAMS to save the log to an external file (in this case `trnspor.log`), instead of writing it to the screen. In order to get an idea about the data at this point, we use the `GDXDUMP` tool to display the contents of the GDX file by running the following command:

```
gdxdump trnspor.gdx symbols
```

By adding the `GDXDUMP` option `symbols`, we will only display the table of contents (shown below) rather than all data stored in `trnspor.gdx`.

```
* GDX dump of trnspor.gdx
* Library in use : C:\GAMS23.3
* Library version: GDX Library Nov 1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
* File version : GDX Library Nov 1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
* Producer : GAMS Base Module Nov 1, 2009 23.3.3 WIN 14929.15043 VIS x86/MS Windows
* File format : 7
* Compression : 0
* Symbols : 12
* Unique Elements: 5
  Symbol Dim Type Explanatory text
  1 a 1 Par capacity of plant i in cases
  2 b 1 Par demand at market j in cases
  3 c 2 Par transport cost in thousands of dollars per case
  4 cost 0 Equ define objective function
  5 d 2 Par distance in thousands of miles
  6 demand 1 Equ satisfy demand at market j
  7 f 0 Par freight in dollars per case per thousand miles
  8 i 1 Set canning plants
  9 j 1 Set markets
 10 supply 1 Equ observe supply limit at plant i
 11 x 2 Var shipment quantities in cases
 12 z 0 Var total transportation costs in thousands of dollars
```

Once we have a GDX file, we can use `GDX2ACCESS` to create a MDB or an ACCDB file. Versions of MicroSoft Office prior to version 2007 use the file extension `.mdb`, while version 2007 and later versions use the file extension `.accdb`. We write all the data stored in the GDX file `trnspor.gdx` to `trnspor.mdb` resp. `trnspor.accdb` by running the following code:

```
gdx2access trnspor.gdx
```
Note that we do not referenced an INI file in the previous command in order to specify the options, i.e. 
GDX2ACCESS will consult the file gdx2access.ini located in the same directory as gdx2access.exe or, if the 
file does not exist, the tool will continue using the default settings (see Options). The information written 
to the log is shown below, including the identifiers of the symbols which are written to trnsport.accdb and 
the elapsed time per symbol and in total.

GDX Access ALFA 23Mar10 23.4.0 WIN 16693.16738 VS8 x86/MS Windows
Creating C:\GAMS support\settext\temp1.accdb with Access: 0.48 seconds
Using temp directory C:\Users\Paul\AppData\Local\Temp\n  i. Insert: 0.00 seconds
  j. Insert: 0.00 seconds
  a. Insert: 0.00 seconds
  b. Insert: 0.00 seconds
  d. Dump: 0.00 seconds Load: 0.05 seconds
  f. Insert: 0.00 seconds
  c. Dump: 0.00 seconds Load: 0.05 seconds
  x. Dump: 0.00 seconds Load: 0.03 seconds
  z. Insert: 0.00 seconds
  cost. Insert: 0.00 seconds
  supply. Insert: 0.02 seconds
  demand. Insert: 0.02 seconds
Renaming C:\GAMS support\settext\temp1.accdb -> trnsport.accdb
Total elapsed time: 0.92 seconds

The resulting ACCBD, opened with MS Access, is shown in Figure 1 (limited to the parameter c on the 
left and the variable x on the right).

As can be seen, every identifier is stored in its own table. For parameters like c, the value is stored in a 
column named value, while variables like x and equations have the columns level, marginal, lowerbound 
and upperbound. A possible additional field (scale for NLP’s, priority for MIP’s, stage for stochastic 
problems) is not exported. If needed, you can assign such a quantity to a parameter in GAMS before 
writing the GDX file. The complete process shown here can be automated as demonstrated in Example 1.

If no domain information is available for a symbol (like B in the code below), each index position gets a 
column and will be labeled automatically with dim1, dim2, etc. If domain information is available, the 
columns will use that information but keeping the names unique (like A in the code below). The small 
example below shows how the index positions are made unique using the (relaxed) domain information by 
adding an ascending number to the identifier.

Set i / i1*i5 /;
Alias (i,j,k);
Parameter A(i,i,i) 'domain informations, but the column name would not be unique';
A(i,j,k) = uniform(0,1);
Parameter B 'no domain information' / i1.i1 1, i1.i2 2, i2.i1 3, i2.i2 4 /;
execute_unload 'AB.gdx', A, B;
execute '=gdx2access AB.gdx';

The resulting database file, opened with MicroSoft Access, is shown in Figure 2 (parameter A on the left, 
parameter B on the right).
6.14.4.2 Example 1 - Dumping the Contents of trnsport.gdx

This example will solve the [trnsport] model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Access and MS Access is launched to inspect the results. This is a small example that should run very quickly.

execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2access trnsport.gdx';
$libInclude win32 shellExecute trnsport.accdb

Note: the equal signs in front of the external programs indicate we don’t go through a shell (e.g. command.com or cmd.exe). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the ‘=’, such errors go undetected and the GAMS model will continue.

The command $libInclude win32 shellExecute trnsport.accdb will launch Access to view the AC-CDB file. This assumes that the version of Access installed is a version as recent as version 2007. Earlier versions will generate a database with the extension .mdb and the shellExecute command needs to be changed as follows:

* view generated file create
$libInclude win32 shellExecute trnsport.mdb

The complete example is also part of the GAMS Data Utilities Library, see model [GDX2ACCESSExample1] for reference.

6.14.4.3 Example 2 - Writing Explanatory Text to Database

In this example, we write a few sets to a GDX file; two of the sets written have explanatory text for set elements. We use the option etFlag to get this text saved in the Access database file along with the corresponding set elements. Without using the option, only the set tuples are saved in the database.

Running this example on a machine with Access 2007 or a later version installed will create a database with the .accdb file extension that cannot be read by an older version of Access. We use the dbVersion option to save the database in the MDB format. Both options must be specified in an user defined INI file within the settings section, see howToWrite.ini in the code below.

```
Set
  i / i1 'one', i2 'two', i3 'three', i4 'four' /
  j / j1*j4 /
  ij(i,j) / i1.j1 'red', i2.(j2,j3) 'green', i3.(j1,j2) 'blue' /;

.gdxOut example2.gdx
$unload i j ij
.gdxOut
$onEcho > howToWrite.ini
[settings]
etFlag=1
dbVersion=10
$offEcho
$call =gdx2access example2.gdx @howToWrite.ini%
```
When we open the database and inspect the table created from the set \( ij \), we see the explanatory texts stored along with the corresponding set elements. The column containing the explanatory text will be labeled with SetText automatically. The column headers can be renamed as demonstrated in Example 5 - Renaming Fields.

The complete example is also part of the GAMS Data Utilities Library, see model [GDX2ACCESSExample2] for reference.

### 6.14.4.4 Example 3 - Dumping a large Table to Database

This is an artificial example where we generate a large identifier in GAMS: a parameter with a million elements. This parameter will be dumped to an MS Access Database afterwards. Note that the GDX2ACCESS execution will last several seconds. The resulting database file has approximately 36MB of size.

```gams
Set i / i1*i1000 /;
Alias (i,j);
Parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'example3.gdx', p;
execute '=gdx2access example3.gdx';
```

The complete example is also part of the GAMS Data Utilities Library, see model [GDX2ACCESSExample3] for reference.

### 6.14.4.5 Example 4 - Special Value Mapping

To store special values like +INF, -INF, EPS, NA and UNDF in a numeric field in the database, GDX2ACCESS uses a mapping. This mapping can be changed by using an INI file (for the default values, see Options resp. the comments in the second code below). We will define a scalar for each special value in GAMS in order to demonstrate the scalarTable option in addition. By default, every scalar will be written to a new table. By activating the scalarTable option in the INI file, all scalars will be stored together in a single table.

```gams
$onEcho > howToWrite.ini
[settings]
inf=1
mininf=2
eps=3
na=4
undf=5
scalarTable=1
$offEcho
```

The usage of our previously customized INI file howToWrite.ini is indicated by the argument `@howToWrite.ini` within the `execute` statement.
GDX2ACCESS will use names like i, j, dim1, dim2, value etc. for the column headers in the resulting database file. In some cases, this may not be convenient, e.g. when more descriptive field names are required. In the following model, we will show how a small script in VBScript[1] can handle this task. The script will rename the columns i, j, and Value in table c to ifrom, jto, and transportcost. At first, the data is defined and dumped to an Microsoft Access database. The VBscript named access.vbs is written at compile time and later executed at execution time in order to rename the column headers.

Set
   i 'canning plants' / seattle, san-diego /
   j 'markets' / new-york, chicago, topeka /

Parameter
   a(i) 'capacity of plant i in cases'
      / seattle 350
      san-diego 600 /
   b(j) 'demand at market j in cases'
      / new-york 325
      chicago 300
Table d(i,j) 'distance in thousands of miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
\[ c(i,j) = f \times d(i,j)/1000; \]

* export to gdx file.
  execute_unload 'c.gdx', c;

* move to access database
* column names are i and j
  execute '=gdx2access c.gdx';

* rename columns
  execute '=cscript access.vbs';

* view results
  $libInclude win32 shellExecute c.accdb

$onEcho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
set oa = CreateObject("Access.Application")
set oDAO = oa.DBEEngine
Wscript.Echo "DAO Version: " & oDAO.version
Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name
Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name
' rename fields
oTable.Fields.Item("i").name = "ifrom"
oTable.Fields.Item("j").name = "jto"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"
oDB.Close
Wscript.Echo "Done"
$offEcho

The resulting ACCBD, opened with MS Access, is shown in Figure 6 (parameter c before executing the VBScript on the left, parameter c after renaming the column headers on the right).

The complete example is also part of the GAMS Data Utilities Library, see model [GDX2ACCESSExample5] for reference.

6.14.5 References

### 6.15 GDX2HAR and HAR2GDX

GDX2HAR and HAR2GDX are translation Tools for GDX and HAR Files.

**Author**

Mark Horridge, Centre of Policy Studies and the Impact Project, Monash University  
Thomas F. Rutherford, Department of Economics, University of Colorado

**Date**

August, 2003 (revised July, 2007)

**Source of this document:** [http://www.mpsge.org/gdxhar/index.html](http://www.mpsge.org/gdxhar/index.html)

The purpose of this document is to provide an overview of GDX2HAR and HAR2GDX translation tools. The GDX2HAR and HAR2GDX programs efficiently translate between HAR and GDX file formats, and their operation is straightforward. The examples covered in this document have been designed to illustrate potential difficulties which may arise and how these issues can be addressed.

#### 6.15.1 Syntax

**6.15.1.1 Syntax for GDX2HAR**

\[ \text{gdx2har GDX\_file\_prefix [.gdx] [HAR\_file\_prefix [.har]] [/s]} \]

**Comments:**

- `gdx2har.exe` is a Windows console application which runs on 32-bit versions of MSWindows (Win95 or later).
- The program requires `gdxiomh.dll` to be located in the same folder as `gdx2har.exe`.
- When only a single file is specified, the output file is `GDX\_file\_prefix.HAR`.
- Any (and all) single dimensional sets and all parameters in the GDX file are written to the HAR file. Variables, equations and multi-dimensional sets (tuples) in the GDX file are ignored.
- Sets in the GDX file are written as sets in the HAR file. Parameters from the GDX file are written as coefficients in the HAR file.
- The `/s` switch invokes strict enforcement of translation syntax. When this switch is specified, warning messages result in program termination without generation of any output.
- Warning messages are generated when incompatible features are encountered in the translation process, including names or set labels with more than 12 characters or other non-conforming syntax. (Set labels in GEMPACK may not begin with a numeric symbol).

Making sense of `gdx2har` requires some understanding of the underlying file formats. GAMS stores values for parameter matrices in GDX files using a *sparse matrix* format, so that a table like
is stored in a GDX file in a binary format corresponding to the following list:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>USA</th>
<th>France</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agriculture</td>
<td>411</td>
<td>87</td>
</tr>
<tr>
<td>Services</td>
<td>2831</td>
<td>365</td>
</tr>
<tr>
<td>SweetCorn</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Truffles</td>
<td>0</td>
<td>68</td>
</tr>
<tr>
<td>Durian</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The key point is that zero elements are not stored in the GDX format, and nor is explicit information about the domain of parameters as declared in the GAMS program which produced the GDX file. To work around this problem, GDX2HAR uses two strategies to infer the domain of a parameter:

1. GDX2HAR scans the descriptive text of parameters in the GDX file to see if an explicit header and domain have been provided. This information is identified by its enclosure in double square brackets, e.g.

   \[
   \text{PARAMETER OUTPUT}(i,r) \ \text{Base year production} \ [[Y:I*R]]
   \]

   The HAR information enclosed in double square brackets consists of the header key ("Y" in this example) and the domain defined with dimension sets separated by 's. The maximum length of a header key is 4 characters.

2. In the absence of explicit declaration in the descriptive text, GDX2HAR examines the labels of nonzero array elements which are stored in a GDX file to see if they correspond to a declared set. If the row or column set does not correspond to a set stored in the GDX file, a new set is created to define the coefficient domain in the HAR file. In the above example, if the GDX file contained sets defined as

   \[
   \text{GOODS} = \ \{\text{Agriculture, Services, SweetCorn, Truffles, Durian}\}
   \]

   and

   \[
   \text{REGIONS} = \ \{\text{USA, France}\}
   \]

   The GDX2HAR program would then correctly infer that the column set is REGIONS, but it would not conclude that the row set is GOODS. It would instead declare a new set

   \[
   \text{SET1} = \ \{\text{Agriculture, Services, SweetCorn, Truffles}\}
   \]

   with which to dimension the data table in the HAR file. As illustrated in this example, the translation process can be problematic which parameters in the GDX file are not explicitly declared.
6.15.1.2 Syntax for HAR2GDX

```
har2gdx HAR_file_prefix [ .har ] [ GDX_file_prefix [.gdx] ] [/s]
```

Comments:

- `har2gdx.exe` is a Windows console application which runs on 32-bit versions of MSWindows (Win95 or later).
- The program requires `gdxiomh.dll` to be located in the same folder as `gdx2har.exe`.
- When only a single file is specified, the output file is `HAR_file_prefix.GDX`.
- All sets and coefficients in the HAR file are written to the GDX file as sets and parameters, respectively.
- The `/s` switch invokes strict enforcement of translation syntax. When this switch is specified, warning messages result in program termination without generation of any output.

6.15.2 GDX2HAR Examples

6.15.2.1 Example 1: Default operation

This example shows how a GDX file can be written by a GAMS program and the results then transferred into a HAR file. In the small GAMS example two sets and three parameters are defined. The program output is then written to the GDX file. Set and parameter names are shorter than 12 characters and conform to GEMPACK syntax rules, so the translator retains all same names in the HAR file. In this case, set and parameter names have fewer than 4 characters, so the HEADER identifiers in the HAR file are identical to the GAMS names.

GDX2HAR infers domains of the parameters from the array elements, which are all non-zero in this case.

Set
```
i / a, b, c /
j / red, green, blue /;
```

Parameter
```
x 'Scalar value' / 1.5 /
y(i) 'Vector of values' / a 10.2, b 1.3, c 1.5 /
z(i,j) 'Matrix of random values';
```

```
z(i,j) = uniform(0,1);
```

This program may be stored as `ex1.gms` and then run from the command line as

```
gams ex1 gdx=ex1
```

If the program is executed from the GAMS IDE a GDX file can be generated by adding `gdx=ex1` to the Additional Parameters box in the File/Options/Execute dialogue. The resulting GDX file can be opened in the GAMS IDE and appears as follows:

The GDX file can be translated into HAR format with the command:
As specified here, GDX2HAR program output (including warning messages) is written to ex1.log:

Running program: C:\GAMS21.0\GDX2HAR.EXE
Input file: H:\gdxhar\examples\ex1.gdx
Output file: H:\gdxhar\examples\ex1.har
Deleted existing file: H:\gdxhar\examples\ex1.har
Loaded GDX library: C:\GAMS21.0\gdxiomh.dll
Above is DLL version: _GAMS_GDX_V228_2003-05-07
GDX file was produced by:
  GAMS Rev 134 May 1, 2003 WIN.00.NA 21.0 134.000.041.VIS P3PC
  Using GDX library: _GAMS_GDX_V228_2003-05-07
GDX file contains 5 symbols and 6 set elements.
Reading GDX set "i".
Reading GDX set "j".
Reading GDX array "x".
Reading GDX array "y".
Reading GDX array "z".
Finished OK; created file: H:\gdxhar\examples\ex1.har

The resulting HAR file can be examined using the VIEWHAR utility:

Note that the HAR file contains the five items from the source GDX file as well as two additional sets named OGEL and NGEL. These sets are always provided to provide a consistent report in the event that set labels may have been translated.

6.15.2.2 Example 2: Missing sets are constructed.

In this example, sets are not written to the GDX file. GDX2HAR then generates the sets based on the nonzero patterns of the parameters. This example illustrates how a GDX file can be written and translated into HAR format within a GAMS program.

Set
  i / a, b, c /
  j / red, green, blue /

Parameter
  x 'Scalar value' / 1.5 /
  y(i) 'Vector of values' / a 10.2, b 1.3, c 1.5 /
  z(i,j) 'Matrix of random values';

z(i,j) = uniform(0,1);

execute_unload 'ex2.gdx', x, y, z;
execute 'gdx2har ex2 > ex2.log';

The resulting HAR file appears as follows:

The sets titled Set1 and Set2 have been introduced by GDX2HAR and are stored in the HAR file under headers S1 and S2. GDX2HAR has inferred that y and z share the common dimension set1, but does not know that set1 was called "i" in the GAMS program.
6.15.2.3 Example 3: Zeros can create problems.

If one element of the \( w \) vector is zero, GDX2HAR does not assume that the coefficient is defined over set "i". Instead GDXHAR introduces a new set to define the domain.

\[
\begin{align*}
\text{Set } & \text{/ a, b, c /; } \\
\text{Parameter } w(i) \text{ 'Vector of values' / a 10.2, c 1.5 /; } \\
\text{execute_unload 'ex3.gdx', i, w; } \\
\text{execute 'gdx2har ex3 > ex3.log'; }
\end{align*}
\]

Above, the set Set1 contains only 2 members, a and c, corresponding to non-zero elements of \( w \).

6.15.2.4 Example 4: EPS can be used to define a domain.

The GAMS language includes a special value EPS, standing for "epsilon," an infinitesimally small but nonzero number. When EPS is added to every element of an array over a given domain, then when that array is written to the GDX file the zeros become visible. The EPS values which are stored in the GDX file appear as a true zero in the translated HAR output file.

\[
\begin{align*}
\text{Set } & \text{/ a, b, c /; } \\
\text{Parameter } w(i) \text{ 'Vector of values' / a 10.2, c 1.5 /; } \\
\text{execute_unload 'ex3.gdx', i, w; } \\
\text{execute 'gdx2har ex3 > ex3.log'; } \\
\text{w(i) = w(i) + eps;} & \\
\text{execute_unload 'ex4.gdx', i, w; } \\
\text{execute 'gdx2har ex4 > ex4.log'; }
\end{align*}
\]

6.15.2.5 Example 5: Explicit declaration of HAR coefficients.

As an alternative to adding EPS to all parameter values, the header and coefficient domain may be specified in the GAMS declaration of a parameter. The HAR declaration is provided within double square quotes, [[ ]].

In this example the GAMS parameter is 12 characters in length. This name may be used in the HAR file to define the coefficient, but it may not be used as the header. (Headers are limited to four characters.) It is possible to define a specific header and domain within the descriptor text of the GAMS parameter.

\[
\begin{align*}
\text{Set } & \text{/ a, b, c /; } \\
& \text{j / red, green, blue /; } \\
\text{Parameter zs_long_name(i,j) 'Param with header and domain [[z:i*j]]'; }
\end{align*}
\]
6.15 GDX2HAR and HAR2GDX

* Note that missing rows are not a problem when the domain is explicitly specified:
zs_long_name(i,j) = uniform(0,1);
zs_long_name("b",j) = 0;
display zs_long_name;
execute_unload 'ex5.gdx', i, j, zs_long_name;
execute 'gdx2har ex5 > ex5.log';

6.15.2.6 Example 6: Long names are truncated

In GAMS, a set or parameter may have as many as 31 characters. GDX2HAR truncates such names to 12 characters.

Parameter GAMS_name_with_31_characters_OK 'A long GAMS identifier' / 1.5 /;
execute_unload 'ex6.gdx', GAMS_name_with_31_characters_OK;
execute 'gdx2har ex6 > ex6.log';

In this example GDX2HAR generates the following HAR file:

**** Warning: To conform with GEMPACK rules, GDX symbol GAMS_name_with_31_characters_OK was converted to GAMS_name_wi

6.15.2.7 Example 7: The GAMS symbol table controls set sequencing

Attention to the GAMS symbol table is needed if set sequences in the HAR file are to be in a particular order. Consider the following GAMS program:

Set
  r 'Selected South American countries' / ARG, BRA, COL, PER, BOL, URG /
  e 'Selected energy goods' / OIL, COL, GAS, ELE /;
display e;

Parameter d(e,r) 'Energy demands';
d(e,r) = uniform(0,1);
execute_unload 'ex7.gdx', r, e, d;
execute 'gdx2har ex7 > ex7.log';

Note that the listing file output of this program presents set e is a ordered in a sequence which differs from the declared sequence:

---- 5 SET e Selected energy goods

COL, OIL, GAS, ELE
The point is that GAMS orders all output rows and columns in accordance with the sequencing of the *global symbol table*. This aspect of GAMS carries over into HAR file generation. In this example, the order in which set \(e\) is displayed in the listing file is the same as the order in which the rows are sorted in the GDX and HAR files:

One way to control the global symbol table in a GAMS program is to declare a *fictive* set at the top of a program in which set elements are defined in the preferred sequence. For example, in the case of the previous program, the following declaration would produce the desired sequence of both sets \(R\) and \(E\) in the HAR file:

```gams
Set symbols / ARG, BRA, OIL, COL /
  r 'Selected South American countries' / ARG, BRA, COL, PER, BOL, URG /
  e 'Selected energy goods' / OIL, COL, GAS, ELE /
```

This produces a consistent ordering of both sets \(r\) and \(e\), but this type of work-around may not always be possible. (See example 9 below.)

### 6.15.2.8 Example 8: Set elements may be truncated or revised.

Restrictions in GEMPACK on the length of set elements and the use of embedded blanks enforced by GDX2HAR. Also, GEMPACK does not allow for set elements to begin with a digit.

```gams
Set
c / 'New York', 'San Francisco', 'Los Angeles' /
  t / 2000*2010 /
executeUnload 'ex8.gdx', c, t;
execute 'gdx2har ex8 > ex8.log';
```

With this example GDX2HAR produces the following warning message:

```
**** Warning: To conform with GEMPACK requirements,
the following GDX set elements were changed:

  New York became NewYork
  San Francisco became SanFrancisco
  Los Angeles became LosAngeles
  2000 became A2000
  2001 became A2001
  2002 became A2002
  2003 became A2003
  2004 became A2004
  2005 became A2005
  2006 became A2006
  2007 became A2007
  2008 became A2008
  2009 became A2009
  2010 became A2010
Reading GDX set "c".
Reading GDX set "t".
**** Warning: Some GDX set elements were changed.
There were 2 warnings.
```

GDX2HAR also alters identifiers to assure that they remain unique, for example, after truncation to 12 letters.
6.15.3 HAR2GDX Examples

Example 9: GlobalSet is provided by HAR2GDX to sequence the GAMS symbol table.

HAR2GDX constructs a symbol table which, where possible, provides properly sequenced arrays in the resulting GDX file. A simple example illustrates how this works. In the source HAR file there are two sets, COM and HAR. COM consists of \( \text{[Cereals, OtherCrops, Power, Services]} \), and IND contains \( \text{[Agriculture, Nuclear, CoalFired, Services]} \). The HAR file contains a single numeric matrix, MAKE, which appears in VIEWHAR as follows:

When the HAR file \text{ex9.har} (from href="examples.zip") is translated by HAR2GDX, the resulting GDX file contains three sets and one parameter array. The sets include COM and IND, as well as a GlobalSet which is inserted to provide a means of sorting the ata arrays into proper sequence. By virtue of the HAR2GDX global symbol table, the translated MAKE array appears in the GDXVIEWer as follows:

If, however, the global symbol table is ignored when the data is read into a GAMS program, as in the example:

```gams
Set
   COM(*) 'Set of commodities'
   IND(*) 'Set of industries';

$gdxIn make.gdx
$load com ind

Parameter make(com,ind) 'Make matrix';
$load make
display make;
```

Then the program develops a global symbol table in which set IND is ordered differently than in the source HAR file. The reason is that GAMS constructs a symbol table sequentially, so if COM is read before IND, then Services is introduced prior to Agriculture, Nuclear, and CoalFired:

```
---- 11 PARAMETER make Make matrix

<table>
<thead>
<tr>
<th></th>
<th>Services</th>
<th>Agriculture</th>
<th>Nuclear</th>
<th>CoalFired</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cereals</td>
<td>7.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OtherCrops</td>
<td>4.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td>2.000</td>
<td>14.000</td>
<td></td>
</tr>
<tr>
<td>Services</td>
<td>29.000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

On the other hand, if GlobalSet is read first, then the GAMS set order (in this example) is identical to the HAR file order:

```gams
Set
   GlobalSet(*) 'Set provided by GDX2HAR to order symbol table'
   COM(*) 'Set of commodities'
   IND(*) 'Set of industries';

$gdxIn make.gdx
$load globalset com ind

Parameter make(com,ind) 'Make matrix';
$load make
display make;
```
6.15.3.2 Example 10: Some GEMPACK datasets cannot be represented in GAMS without reordering sets.

The following is an example of a HAR file which cannot be translated to GAMS without reordering sets. Within ex10.har the sets IND and COM in this example are, respectively, \([A,B,C,D]\) and \([D,C,B,A]\):

The HAR file can be translated to GDX, but this produces the following error message:

```
***** Warning: Inconsistent order: elements "D" and "C" in set COM
***** Warning: Inconsistent order: elements "C" and "B" in set COM
***** Warning: Inconsistent order: elements "B" and "A" in set COM
```

This message means that HAR2GDX was unable to construct a global symbol table which is ordered consistently with both HAR sets COM and IND. The translated data appears as:

Within the GDX file both COM and IND will be ordered \([A,B,C,D]\), and the elements of \(X\) are displayed in that order. Notice that values for particular array elements are translated correctly, e.g. \(X("A","B") = 14\) in both HAR and GDX files.

6.15.4 Concluding Remarks

GDX2HAR and HAR2GDX efficiently translate between HAR and GDX file formats. Although the examples above concentrate on potential problems, in practice both programs are generally easy to use. To avoid problems:

- if you are using GAMS to prepare a GDX file for translation to HAR, remember that GDX files do not naturally contain information about array domains – the sets over which the array is defined. To assist GDX2HAR, you should store associated sets in the GDX file and include domain information and a suggested header key in the "explicit text" description of each GAMS array declaration, as in example 5 above.

- if you are using GEMPACK to prepare a HAR file for translation to GDX, avoid creating sets which share common elements that are ordered differently. If you follow this rule, the GAMS user can use GlobalSet (prepared by HAR2GDX) to ensure that GAMS orders set elements in the same way as GEMPACK. See example 9 above.

- Try to use names (for arrays, sets and set elements) which are legal in both GEMPACK and GAMS. Identifiers of maximum length 10 with first character one of \([A..Z,a..z]\) and remaining characters in \([A..Z,a..z,0..9]\) will translate most smoothly.
6.16 GDX2SQLITE

A Tool to dump GDX contents into SQLite database file

Author

Erwin Kalvelagen

Date

November 30, 2015

6.16.1 Introduction

GDX2SQLITE.EXE is a tool to dump the complete contents of a GAMS GDX file (see GAMS Data eXchange (GDX)) into a SQLite database file (the website http://www.sqlite.org/ contains a wealth of information on SQLite).

A SQLite database is stored in a single file so it can be easily e-mailed or otherwise transmitted. The main advantages of using SQLite over other single file database systems such as MS Access is that SQLite is free and in the public domain and that it does not impose a 2 GB file size limit. For some large data sets this size limit present in MS Access and DBF database files causes problems. Another useful format is CSV files, but typically several CSV files are needed to store a data set stored in a GDX file. Many programs support reading SQLite database files, either through a native database access driver or via a standard ODBC interface. In summary: SQLite is a useful export format for GAMS solution data sets.

6.16.2 Usage

'GDX2SQLITE' is a command line tool that is best called from within a GAMS program using the $call or Execute statement, e.g.:

execute_unload "results.gdx", yield, price;
execute "gdx2sqlite -i results.gdx -o results.db";

The following options are available:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i gdxinputfile</td>
<td>Specifies the input GDX file. Typically this is a file with a .gdx extension</td>
</tr>
<tr>
<td>-o sqloutputfile</td>
<td>Specifies the output SQLite database. Typically this file has a .db extension.</td>
</tr>
<tr>
<td>-debug</td>
<td>This is an optional flag that will cause gdx2sqlite to print additional debugging information.</td>
</tr>
<tr>
<td>-expltext</td>
<td>This optional flag will export explanatory text for set elements.</td>
</tr>
<tr>
<td>-append</td>
<td>Don’t delete the database file before processing. This will allow adding new symbols in new tables. We will not allow adding data to existing tables.</td>
</tr>
<tr>
<td>-small</td>
<td>Write data strings in a separate table. A user-friendly SQL VIEW is created to hide the complexities of the joins.</td>
</tr>
<tr>
<td>-fast</td>
<td>Try to speed up writing the data using some non-standard pragmas. Using both -small -fast will write the data most efficiently.</td>
</tr>
<tr>
<td>-varchar</td>
<td>String columns will have the type VARCHAR(255) instead of TEXT.</td>
</tr>
</tbody>
</table>
An example of explanatory text is:

```gams
Set cty / AFG 'Afghanistan'
        AGO 'Angola'
        ALB 'Albania' /;
```

In GAMS set elements have a maximum length of 63 characters. Explanatory text has a maximum length of 255 characters.

### 6.16.3 How data is stored

#### 6.16.3.1 Gams issues

GAMS does not store zero values (or default records for variables and equations). Such non-existing records will not be exported to the GDX file and to the database either. To force a zero to be exported, set it to `EPS` in GAMS. E.g.:

```gams
p(i)$\{p(i)=0\} = EPS;
```

In case of doubt you are encouraged to inspect the GDX file.

#### 6.16.3.2 Sets

n-dimensional sets are stored as tables with n text columns. In case the option `-expltext` is used, another column may be added with explanatory text.

<table>
<thead>
<tr>
<th>GAMS</th>
<th>SQLite</th>
</tr>
</thead>
</table>
| Set month / jan, feb, mar / year / 2013, 2014 / date(year,month) / (2013,2014),(jan,feb,mar) /; execute unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db"; | sqlite> select * from month; month --------
jan feb mar
sqlite> select * from year; year -------- 2013 2014
sqlite> select * from date; year month
-------- -------- 2013 jan 2013 feb 2013
mar 2014 jan 2014 feb 2014 mar |
| Set month / jan 'january' feb 'fabruary' mar 'march' /; execute unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db -expltext"; | sqlite> select * from month; month expltext
-------- -------- jan january feb fabruary
mar march |

#### 6.16.3.3 Parameters

n-dimensional parameters will have n index columns plus a value column. Scalars are collected in a separate table.
### 6.16.3.4 Variables and Equations

n-dimensional variables and equations have besides n index columns also columns for the level, the lower and upper-bound and the marginal. Scalars are collected in the tables scalarvariables and scalarequations. Note that \( \text{INF} \) and \( -\text{INF} \) are mapped to \( 1.0\times100 \) and \( -1.0\times100 \). The special value EPS is exported as zero. To be complete: UNDEF, NA and acronyms are exported as NULLs.

<table>
<thead>
<tr>
<th>GAMS</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set i / i1*i4 /; Positive Variable x(i); x.l(i) = uniform(0,1);</td>
<td>sqlite&gt;select * from x; i level lo up marginal</td>
</tr>
<tr>
<td>Variable z; z.m = 1;</td>
<td></td>
</tr>
<tr>
<td>execute_unload &quot;data.gdx&quot;; execute &quot;gdx2sqlite -i data.gdx -o data.db&quot;;</td>
<td></td>
</tr>
</tbody>
</table>

### 6.16.3.5 Fixing up names

A database table is not allowed to have columns with the same name. If a name clash is detected new names may be invented.

<table>
<thead>
<tr>
<th>GAMS</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set i / i1*i4 /; Parameter p(i,i); p(i,i) = 1;</td>
<td>sqlite&gt;select * from p; i i2 value</td>
</tr>
<tr>
<td>execute_unload &quot;data.gdx&quot;; execute &quot;gdx2sqlite -i data.gdx -o data.db&quot;;</td>
<td></td>
</tr>
</tbody>
</table>

### 6.16.3.6 Speeding up writing data

With the -small option we write data in a slightly different format. Instead of using strings for the GAMS indices we write integers. The integers can be looked up in a separate table UEL$ were the GAMS UELs (Unique Elements) are stored. We export an SQL view for each symbol to hide the complexities of the joins needed to replace the integers by strings.

<table>
<thead>
<tr>
<th>GAMS</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set i / i1*i4 /; Parameter p(i,i); p(i,i) = 1;</td>
<td>sqlite&gt;select * from p; i i2 value</td>
</tr>
<tr>
<td>execute_unload &quot;data.gdx&quot;; execute &quot;gdx2sqlite -i data.gdx -o data.db&quot;;</td>
<td></td>
</tr>
</tbody>
</table>
The -fast option will set some SQLite pragmas that can speed up the inserts. Basically they will give up some consistency in case the program crashes, in which case the created database may be invalid.

For more information see: http://yetanothermathprogrammingconsultant.blogspot.com/2014/07/a-little-bit-extra-fine-tuning.html.

6.16.4 SQLite Browsers and compatible software

6.16.4.1 SQLite3.exe

From the distribution on http://www.sqlite.org/download.html a command line tool is available that functions as shell for SQLite. An example session can look like:

C:\projects\impact3\sqlite>sqlite3.exe data.db
SQLite version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE [i]([i] TEXT);
CREATE TABLE [p](i1 TEXT,i2 TEXT,value REAL);
sqlite> select * from p;
i1|i1|1.0
i2|i2|1.0
i3|i3|1.0
i4|i4|1.0
sqlite> .quit
C:\projects\impact3\sqlite>

6.16.4.2 SQLite Studio

A visual front-end can be downloaded from http://sqlitestudio.pl/.

6.16.4.3 SQLite Database Browser

Another visual browser is available from http://sqlitebrowser.sourceforge.net/.

6.16.4.4 SQLite and R

The statistical software R can conveniently use SQLite data, and can be called from a GAMS environment as follows:
Get data from GAMS into R via SQLite

Set cty;
$gdxIn %IMPACTPATH%/GDXs/Sets.gdx
$load cty
display cty;

Set maps / data1 'uniform random data between 0 and 1'
   data2 'uniform random data between 0 and 2' /

Parameter mapdata(cty,maps);
mapdata(cty,"data1") = uniform(0,1);
mapdata(cty,"data2") = uniform(0,2);

execute_unload "mapdata.gdx";
execute "gdx2sqlite -i mapdata.gdx -o %DB% -expltext";
execute "%RPATH% --vanilla < %SCRIPT%";

$onEcho > %SCRIPT%
if (!require(RSQLite)) {
   install.packages("RSQLite", repos="http://cran.r-project.org")
   library(RSQLite)
}

sqlite<-dbDriver("SQLite")
db <- dbConnect(sqlite,"%DB%")
dbListTables(db)
maps<-dbGetQuery(db,"select * from maps")
maps
mapdata<-dbGetQuery(db,"select * from mapdata")
mapdata
$offEcho

6.16.4.5 SQLite and Python

Python has built-in support for SQLite:

import sqlite3
db = sqlite3.connect(“turkey.db”)
c = db.execute("select * from yieldl")
for row in c:
   print row
db.close()
6.16.4.6 SQLite ODBC Driver

ODBC is a database access layer for Windows. It allows many Windows programs that need to talk to databases to do this in a database independent manner. The SQLite ODBC driver can be downloaded from:  http://www.ch-werner.de/sqliteodbc/.

6.16.4.7 SQLite and Excel

Excel can read SQLite database files through ODBC.

6.16.4.7.1 Import as Table  To import a table from a SQLite database perform the following steps:

1. Select the Data tab and choose Get External Data From Other Sources
2. Choose the Data Connection Wizard
3. Choose ODBC DSN
4. Select SQLite3 Datasource
5. Enter the name and path of the database file and press OK.
6. Choose a table from the database.
7. Give this selection a name.
8. Choose Import as Table

The result is a table:

6.16.4.7.2 Import as Pivot Table  The same steps can be used to import as Pivot Table. This way we can easily create summary reports, such as:

6.17 GDX2VEDA

Translates a GDX file into the VEDA format.

6.17.1 Usage

gdx2veda gdx vdd [run]

gdx
  GAMS GDX file
vdd
  VEDA Data Definition file
run
  VEDA Run identifier (optional)

The VEDA data file name and run identifier are either taken from the gdx file name or specified with the run name. Use "token with blanks" if needed.
6.17.2 Examples

This example dumps the gdx symbols:

gdx2veda mygdx

This example prints usage and example:

gdx2veda

To print more detailed help message:

gdx2veda --help

Add .csv to the run name to write in csv format

6.17.3 Detailed Help Message

6.17.3.1 VDD file Summary

[DataBaseName]
myveda

[Dimensions] cube dimensions
long_name tuple_element1 tuple_element2 ...

[DataEntries] data for the cube
long_name gams_name tuple_element1 tuple_element2 ...

[DimensionText] for generating .vde file (only for data in [DataEntries])
gams_set tuple_element1 tuple_element2 ...

[DimensionTextAll] for generating .vde file (also for data not in [DataEntries])
gams_set tuple_element1 tuple_element2 ...

[SubSets] for generating .vds file
sub_name gams_name tuple_element1 tuple_element2 ...

[ParentDimension] defines parent-child structure
parent_tab child_tab1 child_tab2 ...

[ParentDimensionTextAll] .vde file definitions with parent-child structure
2d_gams_set parent_tab child_tab
2d_gams_set child_tab parent_tab

[ParentSubSets] .vds file definitions with parent-child structure
sub_name 2d_gams_set parent_tab child_tab
sub_name 2d_gams_set child_tab parent_tab

[Options]
TupleSeparator "string" use a different separator symbol between tuple elements
ShowAllSeparators don’t squeeze unnecessary separators
RelaxDimensionAll relax strict dimensionality checks in DimensionText(All) sections.
ValueDim n if n=2 write PV/DV value pairs for VEDA
SetsAllowed dim1 dim2 write SetsAllowed specification line to VEDA .vd file
Scenario scenarioSet specify the scenario set; a record with expl text goes to .vde
Format veda/csv specify the format of the data files
Not-0 attribute ... don’t write records with zero values for these attributes

[SpecialValues]
EPS "string" value to be used for EPS
INF "string" value to be used for +INF
MINF "string" value to be used for -INF
NA "string" value to be used for NA
UNDEF "string" value to be used for UNDEF

<myveda>

is usually the application name which will be displayed on the top of the VEDA splash screen. When a new VEDA database is created, a new folder with this name will appear:

...veda\database\mayveda_date_time.

Where data and time are the creation time stamp.

<tab_name>

corresponds to the tabs of your VEDA screen

Lines starting with * and empty lines are ignored. Blanks, commas and tabs are delimiters, blanks before and after delimiters are ignored. Quotes around data items are optional. The input data is NOT case sensitive.

6.17.3.2 Veda Data Definition file

Example of a Veda Data Definition file:

* Transport model

[DataBaseName]
myveda

[Dimensions]
* tab-name indices
Plants i
Warehouses j
Links ii jj

[DataEntries]
* veda_attribute gams_name tab1 tab2 ... for gams index 1, 2, ...
"x(i,j) duals" x.m i Warehouses
Shipments x.l i j
SupplyPrice supply.m i
DemandPrice demand.m j
TransportCost c i j
Distance d ii jj
Supply a i
Demand b j
TotalCost  z.l
SupplyNodes  i  i
DemandNodes  j  j
Rate  f

[Dimensions]
* gams_set tab
  i  i

[DimensionsAll]
* gams_set tab
  j  j

[SubSets]
* sub_name gams_name tab
  ii ic  Plants
  ii id  i

Notes:

The long name from the [Dimensions] section can be used as a macro that expands to the tuples it defines. E.g. "Links" is identical to "ii jj".

In the [DataEntries] section a literal tuple element can be defined as /element/.

When ValueDim=2, the [DataEntries] section can contain X.LM entries, indicating both .L and .M needs to written as a pair.

### 6.18 GDX2XLS

A Tool to convert GDX data to MS Excel spreadsheets

Author

Erwin Kalvelagen

Date

June 10, 2005; December 6, 2006

This document describes the GDX2XLS utility which allows to convert data stored in a GDX file into Excel spreadsheets.

### 6.18.1 Overview
Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only GDXXRW is under active development. Therefore, we strongly recommend using GDXXRW for data exchange with Excel.

GDX2XLS is a tool to dump the complete contents of a GDX file to an MS Excel spreadsheet file (.xlsx or .xls file). Every identifier gets its own sheet in the .XLSX file. Excel 2007 or more recent versions will default to an xlsx file; versions before that default to the .xls file format. For instance when we save the results of the transport model from the model library:

C:\tmp>gamslib trnsport
Model trnsport.gms retrieved

C:\tmp>gams trnsport gdx=trnsport lo=2

C:\tmp>gdxdump trnsport.gdx symbols
Symbol Dim Type Records Explanatory text
 1 a 1 Par 2 capacity of plant i in cases
 2 b 1 Par 3 demand at market j in cases
 3 c 2 Par 6 transport cost in thousands of dollars per case
 4 cost 0 Equ 1 define objective function
 5 d 2 Par 6 distance in thousands of miles
 6 demand 1 Equ 3 satisfy demand at market j
 7 f 0 Par 1 freight in dollars per case per thousand miles
 8 i 1 Set 2 canning plants
 9 j 1 Set 3 markets
10 supply 1 Equ 2 observe supply limit at plant i
11 x 2 Var 6 shipment quantities in cases
12 z 0 Var 1 total transportation costs in thousands of dollars

C:\tmp>

The example shows how we copy the trnsport.gms model from the model library, and then solve it. The option gdx=filename will save the complete symbol table to a GDX file. The option lo=2 tells GAMS to save the log to a file (in this case trnsport.log) instead of writing it to the screen. The gdxdump will display the contents of the GDX file (the option symbols will only display the table of contents, rather than all data)

Once we have a GDX file we can use GDX2XLS to create an .XLSX file:

C:\tmp>gdx2xls trnsport.gdx
GDX2XLS 34.1.0 r644dbd9 Released Jan 29, 2021 WEI x86 64bit/MS Window
Output file: C:\tmp\trnsport.xlsx
C:\tmp>

The resulting XLSX file, opened with MSExcel is shown in Figure 1. The first page is the Table of Contents page with all identifiers sorted alphabetically. When clicking on variable x, the sheet shown in Figure 2 is displayed.

The table of contents can be reached again by clicking on the TOC link in the top left corner. The complete process shown here can be automated as is shown in section Model gdx2xls1: import trnsport.gdx. As can be seen, every identifier is stored in its own sheet. Index positions get a column with labels showing their domains, or dim1, dim2, etc. if domain information is not available. By default scalar quantities are collected in a single sheet called scalar.
6.18 GDX2XLS

6.18.2 AutoFilter

By default the exported tables are organized in AutoFilter tables. This will allow you to easily make selections and sort the results.

It is possible to set filters for different columns. Only the rows that meet the criteria will be shown. The columns used in the filter can be recognized by having a blue arrow instead of a black one in the drop down menu header.

Sorting can also be performed on multiple columns: e.g. first sort on one column, then sort on a second column.

The autofilter generation can be turned off using an option in the .ini file.

6.18.3 Options

6.18.3.1 Default ini file

Options are specified in an .INI file. By default, the file gdx2xls.ini located in the same directory as gdx2xls.exe is consulted. If this file is not available, the program will continue using default settings.

6.18.3.2 Custom ini file

It is also possible to tell the program to use a different .ini file. This is done by using an extra argument of the form @inifile. An example would be:

C:\TMP> gdx2xls myfile.gdx @myinifile.ini

In this case the program will not read gdx2xls.ini located in the same directory as gdx2xls.exe but rather myinifile.ini in the current directory.

The ini file can contain two sections: [settings] and [colors]. A complete ini file with all possible settings looks like:

```
[settings]
inf=INF
mininf=-INF
eps=EPS
na=NA
undf=INDF
scalartable=1
tableformatting=1
toc=1
sorttoc=1
autofilter=1
freezeheader=1
indexformat=
valueformat=

[colors]
header=17
body=19
italics=48

[xmlcolors]
link=#0000FF
header=#9999FF
body=#FFFFCC
italics=#969696
```
6.18.3.3 Settings section

A complete description for the [settings] section is:

<table>
<thead>
<tr>
<th>[settings]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf</td>
<td>Special values may need to be mapped to numeric values so the values can be used in formula’s etc. This setting will specify the value for the GAMS INF quantity. The default is the string INF.</td>
</tr>
<tr>
<td>mininf</td>
<td>This is the mapped value for -INF. The default is -INF.</td>
</tr>
<tr>
<td>eps</td>
<td>This is the mapped value to be used for EPS. The default is EPS.</td>
</tr>
<tr>
<td>na</td>
<td>This is the mapped value to be used for NA. The default is NA.</td>
</tr>
<tr>
<td>undf</td>
<td>This is the mapped value to be used for UNDF. The default is UNDF.</td>
</tr>
<tr>
<td>scalarsheet</td>
<td>When this parameter is set to 1, GDX2XLS will generate a separate sheet to collect scalar parameters, scalar equations and scalar variables. This can reduce the number of sheets created with just a single data item. The name of the sheet is fixed: scalar. By default this option is turned on.</td>
</tr>
<tr>
<td>tableformatting</td>
<td>If this option is turned on, extra table formatting is used (adding colors, etc.) to make the tables look better. If this is not needed, this option can be turned off. Default: tableformatting=0.</td>
</tr>
<tr>
<td>toc</td>
<td>Whether or not to add a {Table of Contents} sheet. Default is to generate such a table.</td>
</tr>
<tr>
<td>sorttoc</td>
<td>Whether or not to sort the table of contents alphabetically. If turned off, the table will be displayed in the order in which the identifiers appear in the GDX file. Default is to sort.</td>
</tr>
<tr>
<td>autofilter</td>
<td>Automatically generate AutoFilter enabled tables in Excel.</td>
</tr>
<tr>
<td>freezeheader</td>
<td>Keep headers fixed so they don’t scroll off the screen.</td>
</tr>
<tr>
<td>indexformat</td>
<td>Custom format for index columns. By default this is an empty string.</td>
</tr>
<tr>
<td>valueformat</td>
<td>Custom format for value columns. By default this is an empty string.</td>
</tr>
</tbody>
</table>

An example of setting special values can be found in section Model gdx2xls4: special value mapping.

6.18.3.4 Colors section

A complete description for the [colors] section is:

<table>
<thead>
<tr>
<th>[colors]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>The colorindex to be used as background for table headers. Default is 17.</td>
</tr>
<tr>
<td>body</td>
<td>The colorindex to be used as background for table bodies. Default is 19.</td>
</tr>
<tr>
<td>italics</td>
<td>The colorindex to be used for the font when writing explanatory text. The default is light grey (color index 48).</td>
</tr>
</tbody>
</table>

The [xmlcolors] section is used to specify colors in the XML file to be generated.

6.18.3.5 Custom formats

The format strings consists of four pieces:

[format for $x>0$];[format for $x<0$];[format for $x=0$];[format for strings]
An example given in the Excel help is:

```
#,###.00_;[Red](#,###.00);0.00;"sales ";
```

The codes used here have the following meaning:

<table>
<thead>
<tr>
<th>Formatting Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># (number sign)</td>
<td>displays only significant digits and does not display insignificant zeros.</td>
</tr>
<tr>
<td>, (comma)</td>
<td>To display a comma as a thousands separator or to scale a number by a multiple of one thousand, include a comma in the number format.</td>
</tr>
<tr>
<td>0 (zero)</td>
<td>displays insignificant zeros if a number has fewer digits than there are zeros in the format.</td>
</tr>
<tr>
<td>_ (underscore)</td>
<td>To create a space the width of a character in a number format, include an underscore, followed by the character. For example, when you follow an underscore with a right parenthesis, such as ), positive numbers line up correctly with negative numbers that are enclosed in parentheses.</td>
</tr>
<tr>
<td>[color]</td>
<td>One of [Black], [Blue], [Cyan], [Green], [Magenta], [Red], [White], [Yellow].</td>
</tr>
<tr>
<td>@ (at sign)</td>
<td>Include an at sign (@) in the section where you want to display any text entered in the cell.</td>
</tr>
</tbody>
</table>

Additional formatting characters include:

<table>
<thead>
<tr>
<th>Formatting Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>? (question mark)</td>
<td>adds spaces for insignificant zeros on either side of the decimal point so that decimal points align when formatted with a fixed-width font, such as Courier New. You can also use ? for fractions that have varying numbers of digits.</td>
</tr>
<tr>
<td>condition</td>
<td>Conditions can be specified as follows: [Red][&lt;=100];[Blue][&gt;100].</td>
</tr>
<tr>
<td>exponent</td>
<td>To display numbers in scientific format, use exponent codes in a section, for example, E-, E+, e-, or e+.</td>
</tr>
</tbody>
</table>

A useful format is:

```
[settings]
valueformat=#.????
```

which aligns numbers on the decimal point and depicts zero's as dots just as the listing file is doing.

### 6.18.4 Examples

#### 6.18.4.1 Model gdx2xls1: import transport.gdx

This example will solve the `transport.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a small example that should run very quickly.
$onText
Test of GDX2XLS. Dumps all symbols of
trnsport.gms to trnsport.xlsx.
$offText

execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2xls trnsport.gdx';
$libInclude win32 shellExecute trnsport.xlsx

Notes: the equal signs in from of the external programs indicate we don't go through a shell (e.g. command.com or cmd.exe). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=' such errors go undetected and the GAMS model will continue.

The command $libInclude win32 shellExecute trnsport.xlsx will launch Excel to view the .XLSX file.

6.18.4.2 Model gdx2xls2: import indus89.gdx

This example will solve the indus89.gms model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a fairly large GDX file, with many identifiers, resulting in many sheets in the workbook.

$onText
Test of GDX2XLS. Dumps all symbols of
indus89.gms to indus89.xlsx. This takes
longer as there is a large number of symbols.
$offText

execute '=gamslib indus89';
execute '=gams indus89 lo=3 gdx=indus89';
execute '=gdx2xls indus89.gdx';
$libInclude win32 shellExecute indus89.xlsx

6.18.4.3 Model gdx2xls3: a large table

This is an artificial example where we generate a large identifier in GAMS: a parameter with as many elements as the number of rows that Excel can handle.

$onText
Test of GDX2XLS. Single symbol with 65536-3=65533 records.
Maximum rows that XLS can handle is 65536; an XLSX file allows for slightly more than a million rows
$offText

Set i / i1*i65533 /;
Parameter p(i);
p(i) = uniform(-100,100);
execute_unload 'test.gdx', p;

execute '=gdx2xls test.gdx';
$libInclude win32 shellExecute test.xlsx

If you create a spreadsheet with too many rows, the XLSX file writer will return OLE error 800A03EC. When generating an XML file, an error will occur when Excel loads the file.
6.18.4.4 Model gdx2xls4: special value mapping

To store special values like INF, EPS, NA in a numeric field in the database, GDX2XLS uses a mapping. This mapping can be changed using an INI file.

```plaintext
$onText
Test of GDX2XLS.
Check special value mapping.
$offText

$onEcho > m.ini
[settings]
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=#NA!
undf=#UNDF!
$offEcho

Parameter p(*) / i1 inf, i2 -inf, i3 eps, i4 na /;
p('i5') = 1/0;
display p;

* save parameter p in p.xlsx
* special values are translated to default values:
execute_unload "p.gdx", p;
execute '=gdx2xls p.gdx';
$libInclude win32 shellExecute p.xlsx

* save parameter p in q.xlsx using new mapping
* INF -> 1.0e100  (numeric)
* -INF -> -1.0e100 (numeric)
* EPS -> 0.0      (numeric)
* NA -> #NA!     (string)
* UNDF -> #UNDF! (string)
* execute_unload "q.gdx", p;
execute '=gdx2xls q.gdx @m.ini';
$libInclude win32 shellExecute q.xlsx
```

Numeric values are important if you want Excel being able to operate on these numbers.

6.18.4.5 Model gdx2xls8: custom format

We use a custom value format to color the different values x<0, x=0, x>0 differently. Also align on the decimal point.

```plaintext
$onText
GDX2XLS example: use of custom format
$offText

$onEcho > mexls.ini
[settings]
```
valueformat=[Blue]#.????;[Red]-#.????;[Green]0.????;[Magenta]
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @mexls.ini';
$libInclude win32 shellExecute mexls.xlsx

6.18.4.6 Model gdx2xls9: custom format 2

This uses the more useful custom format valueformat=#.???? (see Figure 7).

$onText
GDX2XLS example: use of custom format
$offText

$onEcho > align.ini
[settings]
valueformat=#.????
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @align.ini';
$libInclude win32 shellExecute align.xlsx

6.19 GDXCOPY

GDX files from different GAMS version can possibly be incompatible. A current GAMS system can read all older GDX file formats. The GDXCOPY utility provides a mechanism to convert GDX files to a format that different GAMS systems can read.

6.19.1 Usage

gdxcopy option inFile outDir

or

gdxcopy option -Replace inFile

The first form copies the converted files to a directory; the second form replaces the original file(s).

inFile

Single file or a file pattern with .gdx file extension.

outDir

The output directory.

Instead of converting the files explicitly using the GDXCOPY utility, files can also be converted by using the environment variable GDXCONVERT with values V5, V6 or V7. The values specified will be used together with the value of the environment variable GDXCOMPRESS to call GDXCOPY as soon as a GDX file is created.

The values of the environment variables can also be set using the GAMS parameters GDXCONVERT and GDXCOMPRESS.

Option:
## 6.19 GDXCOPY

<table>
<thead>
<tr>
<th>Option</th>
<th>Target format</th>
</tr>
</thead>
<tbody>
<tr>
<td>-V5</td>
<td>Version 5</td>
</tr>
<tr>
<td>-V6U</td>
<td>Version 6 uncompressed</td>
</tr>
<tr>
<td>-V6C</td>
<td>Version 6 compressed</td>
</tr>
<tr>
<td>-V7U</td>
<td>Version 7 uncompressed</td>
</tr>
<tr>
<td>-V7C</td>
<td>Version 7 compressed</td>
</tr>
</tbody>
</table>

A current GAMS system can always handle older GDX file formats. The `GDXCOPY` utility provides a mechanism to convert GDX files to a prior format, so an older GAMS system can read these files.

### Notes:

- Version 7 formatted files were introduced with version 22.6 of GAMS; version 6 formatted files were introduced with version 22.3 of GAMS. Prior versions used version 5.
- Some features introduced in version 7 of the GDX file format cannot be represented in older formats.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Action taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension &gt; 10</td>
<td>Symbol is ignored</td>
</tr>
<tr>
<td>Identifier longer than 31 characters</td>
<td>Truncated to 31 characters</td>
</tr>
<tr>
<td>Unique element longer than 31 characters</td>
<td>Truncated to 31 characters</td>
</tr>
<tr>
<td>Domain of a symbol</td>
<td>Domain is ignored</td>
</tr>
<tr>
<td>Aliased symbol</td>
<td>Symbol is entered as a set</td>
</tr>
<tr>
<td>Additional text for symbol</td>
<td>Additional text is ignored</td>
</tr>
</tbody>
</table>

- The Macintosh Intel-based system (DII) which was introduced with GAMS 22.6 does not support GDX conversion into formats version 6 and version 5.

### 6.19.2 Example

In the example below we convert all GDX files to a compressed format for version 6.

```plaintext
dir
  1,219 t1.gdx
  1,740 t0.gdx
889,973  i.gdx
  1,740 pv.gdx
894,672 bytes

gdxcopy -v6c *.gdx newdir

dir newdir
  1,219 t1.gdx
  1,219 t0.gdx
203,316  i.gdx
  1,219 pv.gdx
206,973 bytes
```
6.20  GDXDIFF

The GDXDIFF tool compares the data of symbols with identical name, type and dimension in two GDX files and writes the differences to a third GDX file. A summary report will be written to standard output.

6.20.1  Usage

gdxdiff file1 file2 {diffile} {options}

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. GDXDIFF requires two parameters, the file names of two GDX files. An optional third parameter is the name of the GDX difference file. Without the third parameter, the difference file will be diffile.gdx in the current directory.

diffile = fileName (default = diffile.gdx)

An optional name of the GDX difference file.

6.20.2  Options

The following options can be used when calling GDXDIFF:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eps</td>
<td>0.0</td>
<td>Epsilon for comparison (absolute).</td>
</tr>
<tr>
<td>RelEps</td>
<td>0.0</td>
<td>Epsilon for comparison (relative).</td>
</tr>
<tr>
<td>field</td>
<td>all</td>
<td>Specify a single subfield (l, m, up, lo, prior, scale) of a variable or equation to be compared.</td>
</tr>
<tr>
<td>fldOnly</td>
<td>disabled</td>
<td>Write variables and equations as parameters for the selected subfield.</td>
</tr>
<tr>
<td>id</td>
<td>all</td>
<td>Define specific identifiers of the GDX files to be compared.</td>
</tr>
<tr>
<td>diffOnly</td>
<td>disabled</td>
<td>Controls if differences of variables and equations will be written as parameters or not.</td>
</tr>
<tr>
<td>cmpDefaults</td>
<td>disabled</td>
<td>Enables the comparison of default values.</td>
</tr>
<tr>
<td>cmpDomains</td>
<td>disabled</td>
<td>Enables the comparison of symbol domains.</td>
</tr>
<tr>
<td>matrixFile</td>
<td>disabled</td>
<td>Enables the comparison of GAMS matrix files in GDX format.</td>
</tr>
<tr>
<td>ignoreOrder</td>
<td>disabled</td>
<td>Ignores UEL order of input files to reduce size of output file.</td>
</tr>
<tr>
<td>setDesc</td>
<td>Y</td>
<td>Control if associated text of matching set elements is compared.</td>
</tr>
</tbody>
</table>

Some more detailed remarks on the options:

Eps = value (default = 0.0)

Absolute difference for comparisons; see also Comparing numeric values. If the difference between two values exceeds Eps, a difference will be reported. The valid range is
**RelEps** = value (default = 0.0)

Relative difference for comparisons; see also Comparing numeric values. If the value of **RelEps** is exceeded, a difference will be reported.

**field** = fieldName (default = all)

The specified subfield is the only field used for deciding if a variable or equation is different. **FieldName** is one of the following: l, m, up, lo, prior, scale or all.

**fldOnly** (disabled by default)

Used in combination with the **field** option; The variables and equations will be written as parameters for the selected subfield. This option cannot be used in combination with **diffOnly** and requires **field** being set to an actual field name but not to all.

**id** = identifier (default = all)

Limits the comparison to one or more symbols; symbols not specified will be ignored. Multiple identifiers can be specified as: id=id1 id=id2 or as id="id1 id2". When using **GDXDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

**diffOnly** (disabled by default)

Differences for variables and equations will be written as parameters; each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option cannot be used in combination with **fldOnly**.

**cmpDefaults**

Enables the comparison of default values. When using **GDXDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

**cmpDomains** (disabled by default)

Enable the comparison of symbol domains. Note that the difference are not listed in particular in the diffile. When using **GDXDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

**matrixFile**

This activates a special mode to compare GAMS matrix files in GDX format. This is mostly done for internal use. When using **GDXDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

**ignoreOrder**

By default, GDXDiff preserves the UEL order of the input files. If this is set, this is disabled so that records in the output file could show up in a different order than in the input files. Doing this, the output file can be reduced in size.

**setDesc** = boolean (default = Y)

Enable or disable the comparison of associated texts for set elements.
Criterion for comparing numeric Values
The use of Eps and RelEps is best described by the code fragment below.

AbsDiff := Abs(V1 - V2);
if AbsDiff <= EpsAbsolute
then
  Result := true
else
  if EpsRelative > 0.0
  then
    Result := AbsDiff / (1.0 + DMin(Abs(V1), Abs(V2))) <= EpsRelative
  else
    Result := false;

Interpreting the Labels in the difffile
Only symbols with the same name, type and dimension will be compared. Tuples with different values are written to the GDX difference file, and a dimension is added to describe the difference using the following labels:

- ins1 indicates that the tuple only occurs in the first file.
- ins2 indicates that the tuple only occurs in the second file.
- dif1 indicates that the tuple occurs in both files; contains the value from the first file.
- dif2 indicates that the tuple occurs in both files; contains the value from the second file.

6.20.3 Examples
6.20.3.1 Compares two GDX Files and writes the Differences to a third GDX File

In the following example, the [transport] model is solved twice with different capacity data. GDX files are saved for each run, and compared afterwards using GDXXDIFF. The shipments variable is loaded into a new variable used for a display statement. We introduce four new unique elements that are used in the difference file.

* solve and write to unmodified.gdx before manipulating the data
solve transport using lp minimizing z;
execute_unload 'unmodified.gdx', a, x;

* manipulate the data and solve again, write to modified.gdx
a('seattle') = 1.2*a('seattle');
solve transport using lp minimizing z;
execute_unload 'modified.gdx', a, x;

eexecute 'gdxdiff unmodified modified diffile > %system.nullfile%';

* Declare symbols to hold the data for differences
Set diff tags / dif1, dif2, ins1, ins2 /;
Variable xdiff(i,j,diff tags);
Parameter adiff(i,diff tags);
execute_load 'diffile' adiff=a, xdiff=x;
display xdiff,
The display statement generates the following output in the listing file:

```plaintext
---- 101 PARAMETER a capacity of plant i in cases
    seattle 420.000, san-diego 600.000
```

```plaintext
---- 101 VARIABLE xdiff.L

<table>
<thead>
<tr>
<th></th>
<th>dif1</th>
<th>dif2</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new-york</td>
<td>50.000</td>
</tr>
<tr>
<td>san-diego</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new-york</td>
<td>275.000</td>
</tr>
</tbody>
</table>
```

Alternatively, one can open the diff file in the GAMS IDE or GAMS Studio to display the differences.

This example is also part of the GAMS Data Utilities Library, see model [GDXDIFFExample16] for reference.

### 6.21 GDXDUMP

**GDXDUMP** is a tool to write scalars, sets, parameters (tables), variables and equations from a GDX file formatted as a GAMS program with data statements to standard output, GMS or CSV files. To write to a file, use the output redirection `>` provided by the operating system or the output option of **GDXDUMP**.

#### 6.21.1 Usage

```
gdxdump filename {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

#### 6.21.2 Options

The table summarizes the options of **GDXDUMP**. Running **GDXDUMP** without any arguments will write the sets, parameters, scalars etc. to standard output formatted as a GAMS program with data statements.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>none</td>
<td>Specify the output filename.</td>
</tr>
<tr>
<td>version</td>
<td>disabled</td>
<td>Writing version information only.</td>
</tr>
<tr>
<td>symb</td>
<td>none</td>
<td>Specify a single identifier to be written from the GDX.</td>
</tr>
<tr>
<td>UelTable</td>
<td>none</td>
<td>Write all unique elements in the GDX to a single set.</td>
</tr>
<tr>
<td>delim</td>
<td>if format=normal: period if format=csv: comma</td>
<td>Specify a delimiter to separate elements in the output.</td>
</tr>
<tr>
<td>decimalSep</td>
<td>period</td>
<td>Specify a decimal separator.</td>
</tr>
<tr>
<td>noHeader</td>
<td>disabled</td>
<td>Suppress writing the header information.</td>
</tr>
<tr>
<td>header</td>
<td>none</td>
<td>Specify a new header when writing to CSV.</td>
</tr>
<tr>
<td>Option</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>noData</td>
<td>disabled</td>
<td>Only write the headers of the symbols.</td>
</tr>
<tr>
<td>CSVAllFields</td>
<td>disabled</td>
<td>Controls the writing of subfields of a variable or equation to CSV.</td>
</tr>
<tr>
<td>CSVSetText</td>
<td>disabled</td>
<td>Controls the writing of set element text to CSV.</td>
</tr>
<tr>
<td>symbols</td>
<td>disabled</td>
<td>Generate an alphabetical list of all symbols in the GDX file.</td>
</tr>
<tr>
<td>symbolsAsSet</td>
<td>disabled</td>
<td>Write some basic information for all symbols in the GDX as set.</td>
</tr>
<tr>
<td>symbolsAsSetDI</td>
<td>disabled</td>
<td>Write some basic information including domain information for all symbols in the GDX as set.</td>
</tr>
<tr>
<td>domainInfo</td>
<td>disabled</td>
<td>Generate an alphabetical list of all symbols in the GDX file that includes domain information.</td>
</tr>
<tr>
<td>setText</td>
<td>disabled</td>
<td>Show the set text (aka the associated text) in the GDX file.</td>
</tr>
<tr>
<td>format</td>
<td>normal</td>
<td>Specify the output file format.</td>
</tr>
<tr>
<td>dFormat</td>
<td>normal</td>
<td>Specify the numerical format in the output file.</td>
</tr>
<tr>
<td>cDim</td>
<td>N</td>
<td>Controls the writing of the column header when using the CSV format.</td>
</tr>
<tr>
<td>filterDef</td>
<td>Y</td>
<td>Controls the writing of default values.</td>
</tr>
<tr>
<td>EpsOut</td>
<td>EPS</td>
<td>String to be used when writing the value for 'Epsilon'.</td>
</tr>
<tr>
<td>NaOut</td>
<td>NA</td>
<td>String to be used when writing the value for 'Not Available'.</td>
</tr>
<tr>
<td>pInfOut</td>
<td>+Inf</td>
<td>String to be used when writing the value for 'Positive Infinity'.</td>
</tr>
<tr>
<td>mInfOut</td>
<td>-Inf</td>
<td>String to be used when writing the value for 'Negative Infinity'.</td>
</tr>
<tr>
<td>UndfOut</td>
<td>Undf</td>
<td>String to be used when writing the value for 'Undefined'.</td>
</tr>
<tr>
<td>ZeroOut</td>
<td>0</td>
<td>String to be used when writing the value for 'Zero'.</td>
</tr>
</tbody>
</table>

Some more detailed remarks on the options:

output = fileName (default=none)

Write the output to a file. The .gms resp. the .csv file extension must be added when writing to GMS resp. CSV.

This option is demonstrated in Writing GDX to CSV.

-version (disabled by default)

Synonym: -v

Write version information only and terminate; all other options will be ignored.

symb = identifier (default=none)

Specify a single identifier in the GDX file to be written to standard output, GMS or CSV. When writing to CSV, one must specify the symb option.

This option is demonstrated in Writing GDX to standard output and to GMS.

UelTable = identifier (default=none)
Write all unique elements found in the GDX file to a set using the identifier specified as the name for the set.

This option is demonstrated in Writing GDX to standard output and to GMS.

**delim** = [period, comma, tab, blank, semiColon] (if format=normal: default=period, if format=csv: default=comma)

Selects a different delimiter to separate unique elements; period is the default when writing to standard output or GMS, while comma is the default when writing to CSV.

This option is demonstrated in Writing GDX to CSV.

**decimalSep** = [period, comma] (default=period)

Specify a decimal separator.

This option is demonstrated in Writing GDX to CSV.

**noHeader** (disabled by default)

Suppress the header information when writing a single symbol; only the data for the symbol will be written, not its declaration. The option is ignored when writing all symbols. When writing to CSV, the header row can be suppressed by enabling noheader.

This option is demonstrated in Adding double Quotes to an user defined Header when writing to CSV.

**header** = string (default=none)

The string supplied replaces the default header written by the program to a CSV file. If an empty header is desired, the string can be empty; such a string can be written using two single quotes (header=""), while in general it is best to enclose the string with double quotes.

This option is demonstrated in Writing GDX to CSV and Adding double Quotes to an user defined Header when writing to CSV.

**noData** (disabled by default)

Only write the headers for the symbols; no data is written. The option is ignored when writing to CSV.

**CSVAllFields** (disabled by default)

When writing a variable or equation to CSV, all fields (level, marginal, lower, upper, and scale) will be written. Without this option, only the level will be written. When writing a set the option control the writing of the element text.

This option is demonstrated in Writing GDX to CSV.

**CSVSetText** (disabled by default)

When writing a set to CSV, the set element text will be written as the last column in the CSV file in addition to the set elements. Without this option, only the set elements will be written.

This option is demonstrated in Writing GDX to CSV.
symbols (disabled by default)

Generate an alphabetical list of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in Writing GDX to standard output and to GMS.

symbolsAsSet (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, explanatory text) of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in Writing GDX to standard output and to GMS.

symbolsAsSetDI (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, domain information) of all symbols in the GDX file (not valid when writing to CSV).

domainInfo (disabled by default)

Generate an alphabetical list of all symbols in the GDX file that includes domain information (not valid when writing to CSV). The column DomInf can have the following values:

N/A - The function to get the type of domain information is not available
None - No domain was specified (domain is the universe)
Relaxed - The domain is relaxed, i.e. the identifiers shown do not necessarily represent one dimensional sets
Regular - Regular domain; the identifiers shown are one dimensional sets

This option is demonstrated in Writing GDX to standard output and to GMS.

setText (disabled by default)

Show the set text (aka the associated set text or the set element text) in the GDX file (not valid when writing to CSV).

GDX allows a string of text to be associated with each element of a set. The universe of such strings stored for use in any particular GDX file (i.e. the set text list) can be shown with this option.

format = [normal, gamsbas, CSV] (default=normal)

Change the output format and the symbols written.

When using the gamsbas format, the program will not write the declarations for the symbols and only write the Level and Marginal assignment statements for the variables, and the Marginal assignment statements for equations.

The CSV format adds column headers to the output. By enabling the cDim option the unique elements of the last dimension of the symbol will be used as column headers for the values. If domain information is available, the column headers will be made unique if overlapping names have been used for the names of the index positions. If no domain information is available, the index names used will be of the form dim1, dim2, ...

In order to run GDXDUMP, one must specify a single symbol using the symb option when writing to CSV.

The gamsbas and CSV format is demonstrated in Writing GDX to standard output and to GMS resp. Writing GDX to CSV.
**dFormat** = [normal, exponential, hexBytes] (default=normal)

Specify the numerical format in the output file.

**cDim** = [Y, N] (default = N)

Can be used when writing a CSV file; when enabled, the unique elements of the last dimension will be used as column headers for the values.

This option is demonstrated in Writing GDX to CSV.

**filterDef** = [Y, N] (default = Y)

When enabled, default values will be filtered and not written. This option is enabled by default. For example, if the Level field (.L) of a variable is zero, the value will not be written.

**EpsOut** = string (default = EPS)

String to be used when writing the value for 'Epsilon'.

**NaOut** = string (default = NA)

String to be used when writing the value for 'Not Available'.

**pInfOut** = string (default = +Inf)

String to be used when writing the value for 'Positive Infinity'.

This option is demonstrated in Writing GDX to CSV.

**mInfOut** = string (default = -Inf)

String to be used when writing the value for 'Negative Infinity'.

**UndfOut** = string (default = Undf)

String to be used when writing the value for 'Undefined'.

**zeroOut** = string (default = 0)

String to be used when writing the value for 'Zero'.

Note that GDX files can also be viewed using the GAMS IDE or GAMS Studio.

### 6.21.3 Examples

To demonstrate the features of GDXDUMP in the Writing GDX to standard output and to GMS and Writing GDX to CSV examples, we execute the model transport.gms ([TRNSPORT] from GAMS Model Library) initially to create the GDX file transport.gdx by using the GDX command line option. Afterwards, some important features of GDXDUMP are demonstrated.

```plaintext
gams transport gdx=transport
```

The figure shows the symbol listing of the GDX file in total and the variable x in particular, since those data is used quite often in the following examples:

All subfields of the variable x are shown in the next figure:
6.21.3.1 Writing GDX to standard output and to GMS

In this example, we will demonstrate the effect of some basic features of GDXDUMP.

While we write to standard output in the most sections of this example, one can also write the GDX file contents into a GAMS file using the following command to redirect the output:

```
gdxdump trnsport.gdx > GDXContents.gms
```

Alternatively, the output option can be used to specify an output filename.

**Option symbols**

This listing shown above can also be written to standard output by using the symbols option of GDXDUMP:

```
gdxdump trnsport symbols
```

The GDXDUMP program writes the following output (except for the numbering and the records column, the content is identical compared to the figure):

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Dim</th>
<th>Type</th>
<th>Records</th>
<th>Explanatory text</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>Par</td>
<td>2</td>
<td>capacity of plant i in cases</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>Par</td>
<td>3</td>
<td>demand at market j in cases</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>Par</td>
<td>6</td>
<td>transport cost in thousands of dollars per case</td>
</tr>
<tr>
<td>cost</td>
<td>0</td>
<td>Equ</td>
<td>1</td>
<td>define objective function</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>Par</td>
<td>6</td>
<td>distance in thousands of miles</td>
</tr>
<tr>
<td>demand</td>
<td>1</td>
<td>Equ</td>
<td>3</td>
<td>satisfy demand at market j</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>Par</td>
<td>1</td>
<td>freight in dollars per case per thousand miles</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>Set</td>
<td>2</td>
<td>canning plants</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>Set</td>
<td>3</td>
<td>markets</td>
</tr>
<tr>
<td>supply</td>
<td>1</td>
<td>Equ</td>
<td>2</td>
<td>observe supply limit at plant i</td>
</tr>
<tr>
<td>x</td>
<td>2</td>
<td>Var</td>
<td>6</td>
<td>shipment quantities in cases</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>Var</td>
<td>1</td>
<td>total transportation costs in thousands of dollars</td>
</tr>
</tbody>
</table>

**Option domainInfo**

We use the domainInfo option to write some basic information including domain information (column DomInf) to standard output:

```
gdxdump trnsport domainInfo
```

The GDXDUMP program writes the following output:

<table>
<thead>
<tr>
<th>SyNr</th>
<th>Type</th>
<th>DomInf</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Par</td>
<td>Regular</td>
<td>a(i)</td>
</tr>
<tr>
<td>4</td>
<td>Par</td>
<td>Regular</td>
<td>b(j)</td>
</tr>
<tr>
<td>7</td>
<td>Par</td>
<td>Regular</td>
<td>c(i, j)</td>
</tr>
<tr>
<td>10</td>
<td>Equ</td>
<td>None</td>
<td>cost</td>
</tr>
<tr>
<td>5</td>
<td>Par</td>
<td>Regular</td>
<td>d(i, j)</td>
</tr>
<tr>
<td>12</td>
<td>Equ</td>
<td>Regular</td>
<td>demand(j)</td>
</tr>
<tr>
<td>6</td>
<td>Par</td>
<td>None</td>
<td>f</td>
</tr>
<tr>
<td>1</td>
<td>Set</td>
<td>None</td>
<td>i(*)</td>
</tr>
<tr>
<td>2</td>
<td>Set</td>
<td>None</td>
<td>j(*)</td>
</tr>
<tr>
<td>11</td>
<td>Equ</td>
<td>Regular</td>
<td>supply(i)</td>
</tr>
<tr>
<td>8</td>
<td>Var</td>
<td>Regular</td>
<td>x(i, j)</td>
</tr>
<tr>
<td>9</td>
<td>Var</td>
<td>None</td>
<td>z</td>
</tr>
</tbody>
</table>
Check the **domainInfo** option explanation linked above to get a full list of possible values for **DomInf** and their interpretation. In this example, the symbol dimensions were not specified (**none**) or they are defined on the one dimensional sets i and j (**regular**).

**Option symbolsAsSet**
Using the **symbolsAsSet** option one can write the information displayed by enabling the **symbols** option to a set declaration:

```plaintext
gdxdump trnsport symbolsAsSet
```

The **GDXDUMP** program writes the following:

```plaintext
alias (Symbol, Dim, Type, *)
set   gdxitems(Symbol,Dim,Type) Items in the GDX file /
  "i".1."Set" "canning plants",
  "j".1."Set" "markets",
  "a".1."Par" "capacity of plant i in cases",
  "b".1."Par" "demand at market j in cases",
  "d".2."Par" "distance in thousands of miles",
  "f".0."Par" "freight in dollars per case per thousand miles",
  "c".2."Par" "transport cost in thousands of dollars per case",
  "x".2."Var" "shipment quantities in cases",
  "z".0."Var" "total transportation costs in thousands of dollars",
  "cost".0."Equ" "define objective function",
  "supply".1."Equ" "observe supply limit at plant i",
  "demand".1."Equ" "satisfy demand at market j"
/;
```

**Option symb**
One can write a single identifier from the GDX using the **symb** option:

```plaintext
gdxdump trnsport.gdx symb=x
```

The default output includes the declaration header and a data statement when writing a set or parameter. When writing a variable, the output includes the level and marginal subfields:

```plaintext
positive Variable x(i,j) shipment quantities in cases /
  'seattle'.'new-york'.L 50,
  'seattle'.'chicago'.L 300,
  'seattle'.'topeka'.M 0.036,
  'san-diego'.'new-york'.L 275,
  'san-diego'.'chicago'.M 0.00900000000000001,
  'san-diego'.'topeka'.L 275 /;
```

Note the default values are not written (for instance, zero values of the level or lower bound subfields are missing).

**Option UelTable**
Using the **UelTable** option one can write a set (named **allUELs** in this example) containing all unique elements of the GDX file:

```plaintext
gdxdump trnsport output=allUELs.gms UelTable=allUELs
```
The following set declaration will be part of the output file allUELs.gms:

Set allUELs /
    'seattle' ,
    'san-diego' ,
    'new-york' ,
    'chicago' ,
    'topeka' /;

Format gamsbas
Using the format option one can write a variable or equation in the gamsbas format, i.e. there will be no declaration of the symbol and the level and marginal subfields will be written as assignments.

gdxdump transport format=gamsbas

**GDXDUMP** writes the following to standard output:

x.L ('seattle'.'new-york') = 50 ;
x.L ('seattle'.'chicago') = 300 ;
x.M ('seattle'.'topeka') = 0.036 ;
x.L ('san-diego'.'new-york') = 275 ;
x.M ('san-diego'.'chicago') = 0.00900000000000001 ;
x.L ('san-diego'.'topeka') = 275 ;

z.L = 153.675 ;
cost.M = 1 ;
supply.M ('seattle') = Eps ;
demand.M ('new-york') = 0.225 ;
demand.M ('chicago') = 0.153 ;
demand.M ('topeka') = 0.126 ;

### 6.21.3.2 Writing GDX to CSV

Next we specify CSV as the output format and demonstrate some simple features available when writing to CSV.

**Writing to CSV**

One can write to a CSV file by running the following command:

```
gdxdump transport.gdx output=varX.csv symb=x format=csv
```

By doing this, the level subfield of the variable \( x \) is written as a list to the file \( \text{varX.csv} \) specified in the output statement. The header row contains the domain set identifiers \( i \) and \( j \) of \( x \) and the field "Val":

Option cDim
Using the cDim option one can write the data as a table since the right most dimension is used as the column header:

```
gdxdump transport.gdx symb=x format=csv cDim=y
```

We did not specify an output file in this case, so the data will be written to standard output formatted as CSV:

The **GDXDUMP** program writes the following:

```
"i","new-york","chicago","topeka"
"seattle",50,300,0
"san-diego",275,0,275
```

Note the elements new-york, chicago and topeka are written to the header row, since they are elements of the most right dimension j.

**Customizing the output: CSVAllFields, header, decimalSep, delim, pInfOut etc.**
In this section, all subfields of the variable x are written to CSV and the representation is customized. The writing of subfields is enabled by **CSVAllFields**. In addition, the **header** option is used to overwrite the default header (as seen in the two previous paragraphs). For instance, we want to replace i with the term 'canning plants', j with the term 'markets' and 'Val' with the term 'shipment quantities in cases', while we are fine with the default column labels for the level, marginal etc. subfields, but they will be overwritten, too. Therefore, we must specify them again manually in the **header** option. For some reason, the field separator is changed from comma (default) to semicolon and the decimal separator from period (default) to comma using the **delim** and **decimalSep** options. One can define the string written to CSV for special values of GAMS like +INF by using the corresponding option **pInfOut**, which might be useful for further processing of the CSV file.

```
gdxdump transport format=csv output=varX.csv symb=x CSVAllFields header="canning plants;markets;shipment quantities in cases;Level;Marginal;Lower;Upper;Scale" delim=semiColon decimalSep=comma pInfOut=1E+100
```

**GDXDUMP** creates the following output:

```
canning plants;markets;shipment quantities in cases;Level;Marginal;Lower;Upper;Scale
"seattle";"new-york";50;0;0;1E+100;1
"seattle";"chicago";300;0;0;1E+100;1
"seattle";"topeka";0;0,036;0;1E+100;1
"san-diego";"new-york";275;0;0;1E+100;1
"san-diego";"chicago";0;0,00900000000000001;0;1E+100;1
"san-diego";"topeka";275;0;0;1E+100;1
```
Note that the fields in the header are not enclosed by double quotes. This problem is addressed in the next section Adding double Quotes to an user defined Header when writing to CSV.

**Customizing the output: CSVSetText**

The set element text is the value of a set element. Often this is left out when defining sets, but for descriptive models often set element text is provided. When dumping a set to a CSV file the set element text is not written to the CSV file by default. The option CSVSetText enables the writing of the text. The model mexss.gms ([MEXSS] from GAMS Model Library) has a set i of steel plants with set element text:

```gams
Set i 'steel plants' / ahmsa 'altos hornos - monclova'
   fundidora 'monterrey'
   sicartsa 'lazaro cardenas'
   hylsa 'monterrey'
   hylsap 'puebla' /;
```

When we create a GDX file and dump set i as CSV with defaults via the following commands:

```bash
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i header="steel_plants"
```

GDXDUMP creates the following output:

```text
steel_plants
"ahmsa"
"fundidora"
"sicartsa"
"hylsa"
"hylsap"
```

If we add CSVSetText:

```bash
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i CSVSetText header="steel_plants,real_name"
```

GDXDUMP creates the following output:

```text
steel_plants,real_name
"ahmsa","altos hornos - monclova"
"fundidora","monterrey"
"sicartsa","lazaro cardenas"
"hylsa","monterrey"
"hylsap","puebla"
```
6.21.4 Adding double Quotes to an user defined Header when writing to CSV

This section describes a possible workaround for the following problem: Suppose one wish to write a parameter or variable from GDX to CSV, but with a non default header containing double quotes. For this purpose, one might try to use the header option of GDXDUMP to specify an user defined header. Though, there is no way to add double quotes enclosing the single fields using the header option, but this is a quite common standard in CSV files (for instance, if the fields contain reserved characters).

Initially, define the header you want to write to the CSV file and redirect the line to your final CSV file (which will be created at this point, an already existing file will be overwritten!). Note the double quotes enclosing the field content. Adding the single quotes is necessary in order to run the following statement from GAMS IDE (however, they will not be written to the file), while they must be omitted when using the command prompt.

$echo ""canning plants","markets","shipment quantities in cases"" > quotedHeader.csv

Afterwards, call GDXDUMP by specifying the GDX file, the single symbol x we want to write, the output format CSV and redirect the output to the file quotedHeader.csv:

$call gdxdump transport.gdx symb=x format=csv noHeader >> quotedHeader.csv

Appending the output of the recent GDXDUMP call to the already existing file quotedHeader.csv adds the data of x to the file and creates the following CSV file (note that one must prevent GDXDUMP to write the default header by using the command noHeader):

"canning plants","markets","shipment quantities in cases"
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275

6.22 GDXMERGE

The program GDXMERGE combines multiple GDX files into a single GDX file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

6.22.1 Usage

gdxmerge filepattern1 filepattern2 .... filepatternN {options}

Each file pattern represents a file name or a wildcard representation using ? and *. A parameter of the form @filename will process the commands from the text file specified. The result of the GDXMERGE execution will be written to a file called merged.gdx, unless this default is overwritten by the output option.

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.
6.22.2 Options

The following options can be specified:

\[
\text{id} = <\text{ident1}>, <\text{ident2}>, \ldots \text{ (default = none)}
\]

Only merge the symbols ident1, ident2, ...

\[
\text{exclude} = <\text{ident1}>, <\text{ident2}>, \ldots \text{ (default = none)}
\]

Merge all symbols except for ident1, ident2, ...

\[
\text{big} = <\text{integer}>
\]

The size for big symbols.

\[
\text{strict} = \text{true|false} \text{ (default = false)}
\]

The program terminates with an error (non-zero return code) if input files specified cannot be found or a file pattern results in no files. An already existing output file also triggers an error if strict is set to true.

\[
\text{output} = \text{fileName} \text{ (default = merged.gdx)}
\]

The optional output file name.

All symbols with matching type and dimension will be merged. By specifying the parameter id=ident1 the merge process will only be performed for the identifier(s) specified, while exclude=ident1 indicates that all symbols should be merged except for the ones specified in the exclude list. Note that the two options id and exclude are mutually exclusive.

By default, the program reads all GDX once and stores all data in memory before writing the merged.gdx file. The big parameter is used to specify a cutoff for symbols that will be written one at a time. Each symbol that exceeds the size will be processed by reading each GDX file and only process the data for that symbol. This can lead to reading the same GDX file many times, but it allows the merging of large data sets. The formula used to calculate the cutoff is:

\[
\text{dimension} \times \text{totalNumberOfElements}.
\]

The calculated value is doubled for variables and equations.

In addition to the symbols written, a set is added to the GDX file representing all the files processed during the merge operation. The name of the set is Merged_set_1, and is made unique by changing the number. The explanatory text for each set element contains the date and time of the GDX file processed.

Note

- The file merged.gdx, or the file specified with the output parameter, will never be used in a merge operation even if the name matches a file pattern.
- Symbols with dimension 20 cannot be merged, because the resulting symbol will have dimension 21 which exceeds the maximum dimension allowed by GAMS.
6.22.3 Examples

6.22.3.1 Merging several GDX Files

In this example, we solve the [trnsport] model from the GAMS Model Library using different LP solvers. After each run, we write all symbols to a GDX file and merge the files into one file. The variable X is read from the merged file and displayed.

```
$call gamslib trnsport
$call gams trnsport lp=cplex gdx=cplex
$call gams trnsport lp=xpress gdx=xpress
$call gams trnsport lp=conopt gdx=conopt
$call gams trnsport lp=minos gdx=minos
$call gams trnsport lp=snopt gdx=snopt
$call gdxmerge *.gdx

Variable AllX(*,*,*);
$gdxIn merged.gdx
$load AllX=X
$gdxIn

option AllX:5:1:2;
display AllX.L;
```

The display statement generates the following output in the listing file:

```
---- 22 VARIABLE AllX.L  shipment quantities in cases

  seattle  seattle  san-diego  san-diego
 conopt  300.00000  325.00000  275.00000
 cplex   50.00000  300.00000  275.00000  275.00000
 minos   50.00000  300.00000  275.00000  275.00000
 snopt   50.00000  300.00000  275.00000  275.00000
 xpress  300.00000  325.00000  275.00000

Note that the different solutions are combined into a single symbol of a higher dimension. The filenames (resp. the solver used) are added as unique elements.

Instead of using the display statement, we can also use the GAMS IDE or GAMS Studio to display the merged.gdx file. The following figure shows the contents of merged.gdx after selecting the level subfield of the variable to be displayed and arranging the display:

This example is also part of the GAMS Data Utilities Library, see model [GDXMERGEEExample17] for reference.

6.22.3.2 Recursively Merging GDX Files

Some users generate data scenarios for a model in a hierarchical way and store the results in a directory tree: The following example demonstrates how to use three independent scenario scalars a, b, and c with values low, medium, and high to generate 3*3*3=27 scenarios, store these in a directory tree, and merge them recursively. The resulting GDX file has for each scenario scalar a new dimension which makes this data especially suitable for viewing in a pivot table:
2806
----

Tools Manuals
39 PARAMETER p

commodity price

a_high.b_high.c_high.report
a_high.b_high.c_low .report
a_high.b_high.c_med .report
a_high.b_low .c_high.report
a_high.b_low .c_low .report
a_high.b_low .c_med .report
a_high.b_med .c_high.report
a_high.b_med .c_low .report
a_high.b_med .c_med .report
a_low .b_high.c_high.report
a_low .b_high.c_low .report
a_low .b_high.c_med .report
...

new-york

chicago

topeka

0.25
0.20
0.23
0.25
0.20
0.23
0.25
0.20
0.23
0.25
0.20
0.23

0.17
0.14
0.15
0.17
0.14
0.15
0.17
0.14
0.15
0.17
0.14
0.15

0.14
0.11
0.13
0.14
0.11
0.13
0.14
0.11
0.13
0.14
0.11
0.13

The GAMS program that creates the directory structure and runs the model for all 27 scenarios looks as
follows:
set a,b,c;
parameter
ascen(a<) / a_low 0.9, a_med 1.0, a_high 1.1 /
bscen(b<) / b_low 0.9, b_med 1.0, b_high 1.1 /
cscen(c<) / c_low 0.9, c_med 1.0, c_high 1.1 /;
loop((a,b,c),
put_utility ’shell’ / ’mkdir ’ a.tl:0 ’\’ b.tl:0 ’\’ c.tl:0
put_utility ’exec’ / ’gams t.gms lo=2 -idir "%gams.wdir%" -curdir ’ a.tl:0 ’/’ b.tl:0 ’/’ c.tl:0
);
The program creates the directory tree (by creating the leave nodes of the tree), changes into the leave
directory (curDir) and calls GAMS (using iDir to point to the starting location). The program t.gms is
almost an identical copy of the [trnsport] model with the last few lines changed:
* ...
a(i) = %a%*a(i);
b(j) = %b%*b(j);
c(i,j) = %c%*c(i,j);
solve transport using lp minimizing z;
scalar tmodelstat;
tmodelstat = transport.modelstat;
parameter p(j) ’commodity price’;
p(j) = demand.m(j);
execute ’test -d results || mkdir results’;
execute_unload ’results/input’, a, b, c;
execute_unload ’results/report’, x, z, p, tmodelstat;
This code saves the results of a run in to GDX files input.gdx and report.gdx in a results subdirectory.
After the program has executed the 27 runs of t.gms it executes some Python code that walks recursively
through the directory tree and merges GDX files it finds in a results folder in the leaf nodes percolates
the results up in the tree by calling gdxmerge on each high node level. The Python function os.walk
with argument topdown=False ensure the walking of the tree in the correct order. The complete example
can be found in the GAMS Model Library in model [scenmerge]:


import os

# Python code

for root, dirs, files in os.walk(os.path.normpath(r"C:\Users\test\allscen".rstrip()), topdown=False):
    if 'results' in dirs:
        gd = root.count(os.path.sep)
        gdr = root;
        elif not gd == root.count(os.path.sep):
            raise NameError('results subdirs found at different depths in the tree: ' + root + ' and ' + gdr)
            cmd = 'cd "' + root + '" && gdxmerge results' + os.path.sep + '* output=' + root.split(os.path.sep)[-1]
            print('Merging results in ' + root)
            if not 0 == os.system(cmd + '>' + os.devnull):
                raise NameError('problems running: ' + cmd)
            elif root.count(os.path.sep) < gd:
                input = ''
                for d in dirs:
                    if os.path.isfile(os.path.join(root, d, d + '.gdx')):
                        input = input + ' ' + os.path.join(d, d + '.gdx')
                if len(input):
                    cmd = 'cd "' + root + '" && gdxmerge ' + input + ' output=' + root.split(os.path.sep)[-1]
                    print('Merging' + input + ' in ' + root)
                    print(cmd)
                    if not 0 == os.system(cmd + '>' + os.devnull):
                        raise NameError('problems running: ' + cmd)
                    print('All done. Final result file: ' + root.split(os.path.sep)[-1] + '.gdx')

# End of Python code

6.23  GDXMRW

Interfacing GAMS and MATLAB. This document briefly describes GDXMRW (GDX-Matlab Read/Write), a suite of utilities to exchange data between GAMS and MATLAB. The software gives MATLAB users the ability to use all the optimization capabilities of GAMS, and allows visualization of GAMS models directly within MATLAB. The most recent version of GDXMRW is included as part of the current GAMS Distribution.

Author

Steven Dirkse  (sdirkse@gams.com), GAMS Development Corp.
Michael C. Ferris  (ferris@cs.wisc.edu). Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706
Jagdish Ramakrishnan  (jramakrishn2@wisc.edu), Wisconsin Institute for Discovery, University of Wisconsin - Madison, 330 North Orchard Street, Madison, Wisconsin 53715

Date

August 1, 2014

Attention

GDXMRW is deprecated (see GAMS 38 GDXMRW release notes). Please use GAMS Transfer Matlab instead.
6.23.1 Introduction

Optimization is becoming widely used in many application areas as can be evidenced by its appearance in software packages such as Excel and MATLAB. While the optimization tools in these packages are useful for small-scale nonlinear models (and to some extent for large linear models), the lack of a capability to compute automatic derivatives makes them impractical for large scale nonlinear optimization. In sharp contrast, modeling languages such as GAMS and AMPL have had such a capability for many years, and have been used in many practical large scale nonlinear applications.

On the other hand, while modeling languages have some capabilities for data manipulation and visualization (e.g., Rutherford’s GNUPLOT), specialized software tools like Excel and MATLAB are much better at these tasks.

This paper describes a link between GAMS and MATLAB. The aim of this link is two-fold. Firstly, it is intended to provide MATLAB users with a sophisticated nonlinear optimization capability. Secondly, the visualization tools of MATLAB are made available to a GAMS modeler in an easy and extendable manner so that optimization results can be viewed using any of the wide variety of plots and imaging capabilities that exist in MATLAB.

In order to enable this link between GAMS and MATLAB, we have implemented MATLAB callable functions that can efficiently import and export data to and from GAMS through GDX files. The simplest read and write functions, irgdx and iwgdxx, deal with indexed parameters. Without getting into their precise meaning, these parameters are essentially indexed in each dimension by the sequence of integers from 1 to the size of that dimension. As an example of the read function, the following command in MATLAB will store the matrices \( A \) and \( B \) into the caller workspace after reading from the file \( \text{foo1.gdx} \):

\[
>> \text{irgdx('foo1','A','B');}
\]

Similarly, the call

\[
>> \text{iwgdx('foo2','C','D');}
\]

would write the matrices \( C \) and \( D \) (located in the MATLAB workspace) as indexed parameters into a file named \( \text{foo2.gdx} \). Note that, many of the examples found in this manual are also part of the GAMS data utilities models library, referred to here as \text{datalib}. The statements above can be found in datalib example \text{gdxmrw_intro02_init}. For reading and writing more complex data such as sets, variables, equations, and non-indexed parameters, we can use the more sophisticated functions \text{rgdxx} and \text{wgdx}. Further descriptions of these functions together with advanced use of \text{irgdx} and \text{iwgdx} are detailed in the rest of this paper.

In Section Data Transfer, we discuss the data transfer utilities that allow importing and exporting data between MATLAB and GDX files: \text{irgdx}, \text{iwgdx}, \text{rgdxx}, \text{wgdx}. In Section Extended use, we give a few examples of the MATLAB and GAMS interface. In APPENDIX A - Configuring GDXMRW, we provide information about configuring GDXMRW and testing the utilities. In APPENDIX B - Utility functions: \text{gdxWhos} and \text{gdxInfo}, we describe additional utility functions, \text{gdxWhos} and \text{gdxInfo}, that allow viewing contents of GDX files in the MATLAB console. Finally, in APPENDIX C - Calling GAMS model from MATLAB, we discuss the \text{gams} function that with a single call initializes a GAMS model with MATLAB data, executes GAMS on the model, and returns results back into MATLAB.
6.23.2 Data Transfer

This paper describes a suite of tools for exchanging data between GAMS and MATLAB. This data exchange is accomplished via the GDX (i.e. the GAMS Data eXchange) interface and API. There are many advantages to using GDX, including platform independence, space and time efficiency in storing and accessing data, and a guarantee that all GDX data contains no duplicates and is free from any logical or syntax errors that might prevent it from being read into GAMS. The GDX interface is well tested, available to the public, and is the basis for most if not all of the GAMS data utilities.

In this section we discuss four MATLAB routines. The first two, irgdx and iwgdx, are the subject of sections irgdx and iwgdx and are used to quickly and simply read indexed parameters from a GDX file into MATLAB and vice versa. For an example use of these functions, a generic quadratic program function that mimics MATLAB's quadprog function is provided in datalib example gdxmrw_qp3. The rgdx and wgdx routines (sections rgdx and wgdx) are more sophisticated versions of the first two that can read and write more general GDX data, e.g. sets, variables, equations, and non-indexed parameters. To understand the structure of this GDX data, the material in section Indexing with labels (UELs) is essential.

6.23.2.1 irgdx

irgdx is a specialized MATLAB function to do an efficient import of an indexed parameter from a GDX file. The irgdx call will not read non-indexed parameters; it can only read data recognized as an indexed data type. If a GDX file consists of both indexed and non-indexed data, irgdx can still read the indexed data. To see whether particular parameters in a GDX file are indexed, the gdxWhos function described in APPENDIX B - Utility functions: gdxWhos and gdxInfo can be used.

The syntax and functionality of irgdx closely resembles that of the MATLAB load function to import MATLAB formatted data, i.e data stored in MAT files. In Basic syntax and Load symbols into output structure we describe the basic modes of operation where the data read from GDX are either stored in the caller workspace or into a MATLAB structure on the left-hand side of an assignment. In MATLAB sparse form, we describe an optional specification that allows read results to be stored in MATLAB sparse form. Finally, in Subsection Renaming, we describe a simple syntax that allows the renaming of variables/parameters without the need to make expensive copies.

6.23.2.1.1 Basic syntax

The basic syntax for irgdx is

\[
\text{irgdx('gdxFileName', 'sym1', 'sym2', ...);}
\]

The above call will read the indexed parameters named sym1, sym2, and so on from the GDX file whose name is specified in the first argument and store the results in the MATLAB caller workspace. Only the GDX name argument is mandatory. If only this argument is specified (i.e. no symbol names are provided) then all of the indexed parameters stored in the GDX file will be loaded into the workspace. The GDX name can be specified with or without the '.gdx' extension. Note that all of the input arguments must be specified in string form.

As an example of the above syntax, we can read the scalar a0, the vector a1, and the matrix a2 from the file idx1_.gdx with the following command:

\[
\text{>> irgdx('idx1_','a0','a1','a2');}
\]

The three symbols are stored as MATLAB variables a0, a1, and a2 in the caller workspace. If we want to read all the symbols from the file idx1_.gdx, we simply call irgdx with only the file name:

\[
\text{>> irgdx('idx1_');}
\]
6.23.2.1.2 Load symbols into output structure  An alternative to storing the results in the MATLAB caller workspace is to return the results in a MATLAB structure. Here, the syntax is:

\[
\text{s = irgdx('gdxFileName','sym1','sym2',...);}
\]

This call will return the indexed parameters named \text{sym1}, \text{sym2}, and so on in the structure \text{s}, with fields \text{s.sym1}, \text{s.sym2}, and so on containing the values of the respective indexed parameters. The input arguments must all be in string form. Only the GDX name is mandatory; if only this argument is specified, all of the indexed parameters stored in the GDX file will be returned in the output structure. The GDX name can be specified with or without the .gdx extension.

For example, the following call returns the values of parameter a2 in the structure \text{s}:

\[
\text{\texttt{\textgreater \ greater s = irgdx('idx1_','a2');}}
\text{\texttt{\textgreater \ greater s.a2}}
\text{\texttt{\textbar \ ans =}}
\text{\texttt{\textbar \ 50 50}}
\text{\texttt{\textbar \ 50 50}}
\]

To return all the indexed parameters in the fields of the structure \text{s}, we would simply call \text{irgdx} with only the GDX name:

\[
\text{\texttt{\textgreater \ greater s = irgdx('idx1_');}}
\]

See datalib example \text{gdxmrw_irgdx01_init}.

6.23.2.1.3 MATLAB sparse form  By default, \text{irgdx} will store results in MATLAB dense form. However, it is possible to use MATLAB sparse form for parameters with dimension less than or equal to 2; MATLAB’s sparse storage scheme cannot handle matrices with more than 2 dimensions. Using sparse storage will allow for efficient import of especially large sparse indexed parameters. Note that in this context \text{sparse} does not refer to data represented in the form \text{[i,j,...,val]} but rather to MATLAB’s internal sparse storage scheme. To store a particular symbol in MATLAB sparse form, one can simply append ‘:s’ to that symbol’s name in the \text{irgdx} call. For example, the following call will store the parameter a2 into the caller’s workspace in sparse form:

\[
\text{\texttt{\textgreater \ greater irgdx('idx1_','a2:s');}}
\]

Note that if we want to store all the indexed parameters from a specified file in MATLAB sparse form, we would need to specify each symbol individually in order to append the ‘:s’ to each symbol’s name.
6.23.2.1.4 Renaming  The details of this section are rather involved; it can be skipped without loss of continuity.

While an *irgdx* call will typically store results in the caller workspace or create field names in the output structure using the symbol names from the GDX file, it is possible to rename the outputs in MATLAB without making potentially expensive copies. To rename a symbol, we can take advantage of MATLAB’s default copy-on-write mechanism: when an assignment occurs, the values assigned are not actually copied until one of the values (the original or the copy) is actually changed. For example, after executing the following two lines of code, the variable *a2_new* points to the same memory location as *a2* and is not a separate copy of *a2*:

```matlab
>> irgdx('idx1_','a2');
>> a2_new = a2;
```

If however the value of *a2_new* were changed after executing the above lines, MATLAB would then need to make a separate copy, at the cost of memory and time. Now, the command

```matlab
>> clear a2;
```

immediately following the above code would complete the rename of *a2* to *a2_new* as desired. Once *a2* is cleared, we can change *a2_new* without needing to make a copy. Similarly, we can do a rename when we have an output structure:

```matlab
>> s = irgdx('idx1_','a2');
>> s.a2_new = s.a2;
>> s = rmfield(s,'a2');
```

To convince ourselves that this code indeed does not require a copy, we can run the commands in the debug output format. That is, immediately after the assignment of *s.a2* to *s.a2_new* in the above code (before using the `rmfield` function), we can run the following commands:

```matlab
>> format debug;
>> s.a2

ans =

Structure address = 8c1eda0
m = 2
n = 2
pr = 75fa88a0
pi = 0
      50   50
      50   50
```

```matlab
>> s.a2_new

ans =

Structure address = 8c1eda0
m = 2
n = 2
pr = 75fa88a0
pi = 0
      50   50
      50   50
```

Note that the pointer *pr* is the same for both outputs, implying that MATLAB does not make a separate copy prior to a write. A similar test can also be done when parameters are stored in workspace variables rather than a structure.
6.23.2.2 iwgdx

*iwgdx* is a function that creates a GDX file of indexed parameters from MATLAB data. Since *iwgdx* writes only indexed parameters to GDX, the input MATLAB data is simplified: no labels are required or implied. The input matrices can be stored in the usual (dense) scheme or using the MATLAB sparse scheme: *iwgdx* detects and handles the two cases automatically. Note that matrices represented in [i,j,...,val] form are not acceptable as *iwgdx* input.

While the *iwgdx* function is patterned after the *save* function in MATLAB, there are important differences. Firstly, if only the file name is specified, the *save* function stores all variables in the caller workspace, while *iwgdx* creates an empty GDX file. The behavior of *iwgdx* in this case also differs from that of *irgdx*, which reads all parameters when only the file name is specified. Secondly, *iwgdx* has an additional pass by reference and renaming syntax, in which the parameter string name and values are passed as two consecutive arguments. In Subsection Basic syntax, we describe the basic syntax for *iwgdx*. In Subsection Pass by reference syntax, we introduce the pass by reference and renaming syntax. Finally, in Subsection Dimensionality specification, we describe an optional dimensionality specification that allows symbols to be written with a larger dimensionality than the default.

### 6.23.2.2.1 Basic syntax

The basic syntax for *iwgdx* is:

```
iwgdx('fileName','sym1','sym2',...);
```

The above call will write the values of the variables sym1, sym2, and so on in the MATLAB caller workspace into a GDX file with the name given by the first input argument. The file name can be specified with or without the '.gdx' extension. If a GDX file with the same name already exists, it will be overwritten. Note that a call with a single input argument (the GDX name) will raise a warning and will create an empty GDX file. This contrasts to the MATLAB save function, which writes all of the workspace variables when none are specified. In the basic form described here, all of the *iwgdx* input arguments are strings.

To illustrate, suppose the matrices a2 and a3 exist in the MATLAB caller workspace. We can write the results of these matrices into a file test.gdx with the following call:

```matlab
>> iwgdx('test','a2','a3');
```

If the file test.gdx already exists, it will be overwritten. Specifying only the file name will create an empty GDX file:

```matlab
>> iwgdx('test');
Warning: an empty gdx file was created.
```

### 6.23.2.2.2 Pass by reference syntax

When using the basic *iwgdx* syntax, a copy of the workspace variable to be written to GDX is made as part of the call. This is unavoidable: the MATLAB executable (MEX) interface only provides functions that return copies of workspace variables, most likely to avoid overwriting or corrupting data. However, an alternative, more efficient approach is possible by passing the numeric data as an argument. Essentially, the pointer to the data values are passed by reference and no copy is made. This could be especially useful when writing a large amount of data. An additional benefit of this syntax is the ability to write GDX parameters with different names than used in the MATLAB caller workspace. For example:

```matlab
>> iwgdx('test','s1',s1Mat,'s2',s2Mat);
```
In the above command, \( s1 \) and \( s2 \) are strings holding the GDX parameter names, while \( s1\text{Mat} \) and \( s2\text{Mat} \) are matrices containing the values to store. We can also use a mix of the basic syntax with the pass by reference syntax. For example:

\[
\text{>> } \text{iwgdx('test','anew',a,'b','c',c);}\]

This call would save matrix \( a \) as anew in GDX without making a copy, would internally make a copy of \( b \) while saving \( b \) to GDX, and would save \( c \) to GDX without renaming or making a copy.

We can do a timing test to see the performance difference between the basic syntax and the pass by reference syntax. The following commands create a 5000x5000 dense matrix consisting of random values in the workspace.

\[
\text{>> clear; }\]
\[
\text{>> randMat = rand(5000);}\]

First, we do a timing test using the basic syntax:

\[
\text{>> delete test.gdx; } \% \text{ delete file if it exists }\]
\[
\text{>> tic;}\]
\[
\text{>> iwgdx('test','randMat'); }\]
\[
\text{>> t1 = toc }\]
\[
t1 = 5.3346
\]

Now, we do a timing test using the pass by reference syntax:

\[
\text{>> delete test.gdx; } \% \text{ delete file if it exists }\]
\[
\text{>> tic;}\]
\[
\text{>> iwgdx('test','randMat',randMat); }\]
\[
\text{>> t2 = toc }\]
\[
t2 = 5.1636
\]

For the above example, the improvement in time performance is about 3%. Of course, the pass by reference call will also use only the half the memory compared to the basic call. We can expect further improvement in performance time and memory use for even larger data.

\subsection{Dimensionality specification}

It is possible to specify the number of dimensions (i.e. dimensionality) for the symbol written to GDX by appending :\( n \) to the variable/symbol name, where \( n \) is the desired dimensionality of the GDX symbol. This feature is motivated by two quirks of the MATLAB environment. Firstly, all scalars and vectors are stored in MATLAB as 2-dimensional matrices. For example:

\[
\text{>> s0 = 100 }\]
\[
\text{>> s1 = [ 100 ]}\]
results in two variables with identical storage schemes: 1x1 matrices. In the MATLAB environment this isn't a problem: things behave as you would expect them to, and the `isscalar`, `isvector`, and `ismatrix` calls are available to interrogate MATLAB about how it views variables. However, in GDX there is a distinction between a scalar (something with 0 dimensions), a parameter with 1 dimension and length 1, and a 1x1 parameter. We need to be able to create GDX files containing any or all of these.

Secondly, MATLAB "flattens" all variables by removing trailing dimensions whose size is 1. For example, after executing

```matlab
>> B = ones(2,2,2)
>> C = ones(2,2,2,1,1)
```

B and C will be identical 3-dimensional MATLAB arrays, yet we need to be able to create a 2x2x2x1x1 indexed parameter in GDX.

By default, `iwgdx` creates GDX parameters that are consistent with the MATLAB view of the data passed in. For example, executing

```matlab
>> d0 = ones(1,1)
>> d1 = ones(3,1)
>> d2 = ones(3,3)
>> iwgdx('dd','d0','d1','d2')
```

results in indexed parameters with 0, 1, and 2 dimensions, respectively, being written to GDX. However, if we want to write d0 as a vector or matrix or higher-dimensional array, we need to use the optional syntax to specify the dimensionality of the resulting symbol in GDX. For example:

```matlab
>> d0 = pi;
>> p1 = d0;
>> d1 = pi*ones(3,1);
>> p2 = d1;
>> d2 = pi*ones(3,3,1)
>> iwgdx('pp','d0','p1:1','d1','p2:2','d2','p3:3',d2)
```

will write the following symbols to GDX:

- The 0-dimensional symbol `d0`. This is the default behavior, consistent with the MATLAB view that `d0` is a scalar.
- The 1-dimensional symbol `p1`. The scalar variable `p1` was promoted to a 1-dimensional symbol by adding an additional singleton dimension.
- The 1-dimensional symbol `d1`. This is the default behavior, consistent with the MATLAB view that `d1` is a vector.
- The 2-dimensional symbol `p2`. The vector variable `p2` was promoted to a 2-dimensional symbol by adding an additional singleton dimension.
- The 2-dimensional symbol `d2`. This is the default behavior, consistent with the MATLAB view that `d2` is a matrix, i.e. the final singleton dimension is just removed.
- The 3-dimensional symbol `p3`. The matrix variable `d2` was promoted to a 3-dimensional symbol by adding an additional singleton dimension. Independently, call-by-reference was used to rename `d2` to `p3`.

Note that the dimensionality specified must always be a promotion, i.e. the dimensionality of the resulting GDX symbol is only increased from what it would be by default.

See datalib example `gdxmrw_iwgdx01_init`. 
6.23.2.3 Indexing with labels (UELs)

To this point we have been looking at functions to read (irgdx, Section irgdx) and write (iwgdx, Section iwgdx) indexed parameters from and to GDX. Indexed parameters are convenient since their structure is essentially identical to that of MATLAB matrices. This structure can be encapsulated very simply as the number of dimensions and the size or extent of each dimension. Using indexed parameters, we do not need to be very concerned with the structure of the data since little structure exists. However, there is much more to GDX data than indexed parameters. If we want to read and write GDX data in more generality we will need to understand how labels or strings are used to reference GDX data and how these labels are organized within GDX.

In general, GAMS data is referenced with labels instead of with numbers, so that one references demand('chicago') instead of demand(2) in a GAMS model. These labels are also called Unique Element Labels, or UELs. The collection of UELs used in a model or in a GDX file is ordered internally and often referred to as the universe of UELs. For efficiency, it is not necessary to use labels internally in GDX or when using the GDX API. Instead, a correspondence or mapping between integers and labels is established initially and/or built up as labels are introduced, so that integers can be used in place of the UELs that reference the data. It is important to keep in mind that the labels are the key thing in referencing GDX data: when integer maps are used the integers are only used to efficiently represent the labels.

A similar scheme mapping integers to labels is used in the rgdx and wgdx routines for reading and writing general GDX data. In addition to the actual data values being stored, there will be a mapping passed to allow integers to be mapped to labels and vice versa. In some cases, where no mapping is passed and data are written to GDX, a default mapping (1 to '1', 2 to '2', etc.) may be used. When reading, the default mapping to use when passing back data is the universe mapping from the GDX, but it is possible to apply a filter. A filter can reduce the amount of data returned and also change the order of that data.

6.23.2.4 rgdx

rgdx is a MATLAB utility to import data from a GDX file. It takes structural input and returns data back in the form of a structure. This is a very flexible routine as it gives user control over the output data structure. rgdx can read a set/parameter/equation/variable from a GDX file and display results in either full/dense or sparse form. A user can also perform a filtered read to read only certain specific elements of a symbol. It can also perform compression to remove extra zeros.

This routine can take up to two arguments. The first argument is a string input containing the GDX file name. It can be with or without the '.gdx' file extension. If you call this routine with only the GDX file name as an argument then the 'uels' field of output structure will be the global UEL of the GDX file and the rest of the fields of the output structure will be NULL. The second argument is a structure input containing information regarding the desired symbol. The syntax for this call looks like this:

\[
x = \text{rgdx('fileName', structure);}
\]

As an example, we read a 3D parameter, 'test3' from 'sample.gdx'. Here we display this parameter in full format but without redundant zeros:

\[
>> \text{s.name = 'test3';}
>> \text{s.form = 'full';}
>> \text{s.compress = true;}
>> x = \text{rgdx('sample', s)}
\]

\[
x =
\]
name: 'test3'
type: 'parameter'
dim: 3
val: [4x2x2 double]
form: 'full'
uels: {{1x4 cell} {1x2 cell} {1x2 cell}}

>> x.val
ans(:,:,1) =
   3   4
   4   5
   5   6
   6   7

ans(:,:,2) =
   4   5
   5   6
   6   7
   7   8

>> x.uels{1}
ans =
    '1'    '2'    '3'    '4'

>> x.uels{2}
ans =
    'j1'    'j2'

>> x.uels{3}
ans =
    'k1'    'k2'

In the following subsections we will explain the input and output structures. Please note that except for the 'name' and 'uels' fields, all other string fields take case insensitive input. All boolean fields can also be entered as string values as well.

6.23.2.4.1 Input structure To read a symbol from a GDX file we just need to know its name in string format. Thus, the only mandatory field of the input structure is 'name', e.g.

>> s.name = 'test3';

There are several other optional fields of the input structure that give user more control over the output structure. These optional fields are as follows:
1. form
This field represents the form of the output data. Output data can be either in 'full' or 'dense' form or it can be in \([i, j, \ldots, val]\) sparse form. We will label \([i, j, \ldots, val]\) as 'sparse'. A user can enter it as string input with value 'full' or 'sparse'. e.g.

```matlab
>> s.form = 'full';
```

By default the data will be in 'sparse' format. Note that this sparse format differs from MATLAB's internal sparse storage scheme that we referred to in prior sections for the \texttt{irgdx} and \texttt{iwgdx} functions.

2. compress
By default the uels in the output structure will be a global UEL of the GDX file and the 'val' field data will be indexed to this UEL. The \texttt{rgdx} routine allows a user to remove rows and columns with all zeros from the 'val' data matrix and re-indexes the uels accordingly. This is called compression of the data. This can be achieved by setting compress as true in the input structure. Valid values for this field are true and false, either in logical form or in string form. e.g.

```matlab
>> s.compress = 'true';
```

However, we note that compressing the data can be dangerous because the size of the matrix that is read can be incorrect. Essentially, all zero rows and columns are removed, including those that might actually be part of the data values in the symbol matrix.

3. uels
This input field is used to perform a filtered read, i.e. output data matrix will contain values only corresponding to the entered uels. Filtered read is very useful if user just wants certain specific set of data. Uels should be entered in cell array form. It has to be in 1xN form with each column being a cell array representing the uels for that dimension. Each column can have strings, doubles or combinations of both. It also allows a user to enter double data in shorthand notation or a 1 x N matrix. For example, in the previous example we can perform a filtered read to get data corresponding to only the '1\&prime', '3' elements of the first index of the parameter 'test3'. The following function is handy when one needs to generate a UEL listing for the input structure:

```matlab
>> guel = @(s,v) strcat(s,strsplit(num2str(v)));
```

Using the above function, we can create a listing of strings consisting of the string \(s\) appended to each number in the array \(v\). Thus, instead of using the command

```matlab
>> s.uels = {{'1','3'},{'j1','j2'},{'k1','k2'}};
```

we can conveniently use the command

```matlab
>> s.uels = {guel('',[1,3]),guel('j',1:2),guel('k',1:2)};
```

The benefit of using the latter command will be more apparent when creating many elements in the listing. Now, as an example, suppose we would like to do a filtered read on the parameter called \texttt{test2} in \texttt{sample.gdx}. We could use the following commands:

```matlab
>> s.name = 'test3';
>> s.form = 'full';
>> s.compress = false;
>> s.uels = {guel('',[1,3]),guel('j',1:2),guel('k',1:2)};
>> x = rgdx('sample',s)
```

```matlab
name: 'test3'
type: 'parameter'
dim: 3
val: [2x2x2 double]
```
Here it should be noted that we turned off compression while performing the filtered read. This is necessary because the filtered read will give data in accordance with the entered uels and the output uels will be the same as the input uels; thus compression is not possible.

4. field
   This field is required when variables or equations are to be read from a GDX file. Sets and parameters in the GDX file do not have any field value but variables and equations have 5 fields namely, level, marginal, lower, upper, and scale. Thus, it may be useful to enter field as an input when reading an equation or a variable. A user can enter it as a string with valid values being 'l/m/up/lo/s'. e.g.

   >> s.field = 'm';

   By default, the output will be the level value of a variable or an equation.

5. ts
   This represents the text string associated with the symbol in the GDX file. If a user sets this field to be 'true', then the output structure will have one more string field 'ts' that contains the text string of the symbol. e.g.

   >> s.ts = true;

6. te
   GAMS allows a modeler to enter text elements for a set. Similarly to the 'ts' field, if a user sets 'te' to be true in the input structure, then the output structure will contain one more field representing the text elements for that symbol. Please note that text elements only exist for 'sets'. e.g.

   >> s.te = true;

6.23.2.4.2 Output Structure
   As mentioned earlier, output of the rgdx routine will be in structure form. This structure is very similar to the input structure. To get information regarding any symbol, we always need to display its basic characteristics, such as its name, type, value, uels, form, etc. An output structure will always have these fields:

1. name
   It is same as that entered in the input structure name field, i.e., the symbol name in the GDX file.

2. val
   It represents the the value matrix of the symbol. To save MATLAB memory by default it will be in 'sparse' format. e.g.
>> s = rmfield(s, 'form');
>> s

    name: 'test3'
    compress: 0

>> x = rgdx('sample', s)

    name: 'test3'
    type: 'parameter'
    dim: 3
    val: [16x4 double]
    form: 'sparse'
    uels: {{1x8 cell} {1x8 cell} {1x8 cell}}

Here `val` is a 16x4 double matrix. As it is a parameter; thus the last column of the sparse matrix will represent the value and the rest (i.e. the first three columns) will represent its index. Please note that in the case of a `set`, the number of columns in the sparse matrix will be equal to its dimension, i.e., it will not have a column representing its value. Here, the presence of each row in the output `val` field corresponds to the existence of a set element at that index. When a `full` matrix output is specified, a 1 represents existence of a set element and a 0 otherwise.

3. **form**
   It represents the format in which the `val` field is being displayed. As mentioned earlier it can be either in `full` or `sparse` form.

4. **type**
   While reading a symbol from a GDX file it is often very useful to know its type. The `rgdx` routine is designed to read set, parameter, variable and equation. This field will store this information as a string.

5. **uels**
   This represents the unique element listing of the requested symbol in the form of a cell array. It is a 1 x N cell array, where N is the dimension of the symbol. Each column of this array consists of string elements. By default, the output uels will be the same as the global uel of the GDX file, but it can be reduced to element specific local uels if compress is set to be true in the input structure. If a user is using a filtered read, i.e. calling `rgdx` with input uels, then the output uels will be essentially the same as the input uels.

6. **dim**
   It is a scalar value representing the dimension of the symbol.
   Apart from these necessary fields there are a few additional fields as well. They are as follows:

7. **field**
   If we are reading variables or equations, then it becomes useful to know which field we have read, i.e., l/m/up/lo/s. This information is displayed via this field in the form of a string.

8. **ts**
   It display the explanatory text string associated with the symbol. This field only exists in the output structure if the `ts` field is set as `true` in the input structure.

9. **te**
   It is an N dimensional cell array representing the text elements associated with each index of the set. This field only exists in the output structure if the `te` field is set as true in the input structure and the symbol is a set.

See datalib example `gdxmrw_rgdx01_init`. 
6.23.2.5 wgdx

wgdx is a MATLAB routine to create a GDX file containing data from MATLAB. Similar to the rgdx routine, it takes a structure input and can write multiple symbols into a single GDX file with one call. The first argument is the file name of the GDX file to be created in string format; it can be with or without the '.gdx' file extension. The rest of the arguments are structures, each containing data for different symbols to be written into the GDX file. The syntax for the call is:

```matlab
>> wgdx('fileName', s1, s2 ...);
```

If the GDX file already exists in the MATLAB current directory, wgdx will overwrite it; otherwise a new file will be created. After a successful run, it doesn’t return anything back into MATLAB. Most of the fields of its input structures are the same as those of the rgdx output structure. In the example below, we use the wgdx routine to create foo.gdx containing a set ‘l’ and a parameter ‘par’.

```matlab
>> s.name = 'l';
>> s.uels = {{'i1', 'i2', 'i3'}, {'j1', 'j2'}};
>> c.name = 'par';
>> c.type = 'parameter';
>> c.val = eye(3);
>> c.form = 'full';
>> c.ts = '3 x 3 identity';
>> wgdx('foo', s, c)
```

The equivalent code in GAMS to create the above set ‘l’ is:

```gams
Set
  a / i1*i3 /
  b / j1*j2 /
  l(a,b);
l(a,b) = yes;
```

In the next subsection we will explain the input fields in detail.

6.23.2.5.1 Input Structure The necessary fields in an input structure to represent a symbol are given below:

1. **name**
   It is a string representing the name of the symbol.

2. **val**
   It represents the value matrix of the parameter or set. It can be entered in either full or sparse format, whichever is convenient to the user; the corresponding format must be specified in the 'form' field. By default the value matrix is assumed to be in sparse format.

3. **type**
   It is a string input to specify the type of the symbol. The wgdx routine can write a set or parameter into the GDX file. In the previous example, we didn’t specify the type for structure ‘s’ because by default it is assumed to be a set.

4. **form**
   This is a string input representing the format in which the val matrix has been entered. By default it is assumed that the data is specified in sparse format.
5. uels
Similar to the rgdx uels field, this represents the local unique element listing of the symbol in an 1 x N cell array form. Each column of this cell array can contain string or double or both. Again we can make use of a handy function to create a UEL listing as mentioned previously in the section about rgdx:

```matlab
>> guel = @(s,v) strcat(s,strcat(s,strsplit(num2str(v))));
```

As an example, we create a set using this function and the wgdx routine:

```matlab
>> s.name = 'l';
>> s.uels = {guel('i',1:20),guel('j',1:40)};
>> wgdx('foo',s);
```

The above code will create foo.gdx that contains a 2 dimensional set 'l' with set elements for each of the specified uels. If a user would like to create a set with set elements only at specified indices, one could do so by entering the indices in the val field. e.g.

```matlab
>> s.val = [1, 1; 2, 1; 3, 6; 4, 7];
>> wgdx('foo',s);
```

Now the above would result in a set with only elements existing in the indices specified in the val field. If a user enters a structure with only two fields, name and uels, as in the example given in the introduction of this section (structure s), then wgdx will create a full set corresponding to the global uels.

The optional fields are:

6. dim
This field is useful when a user wants to write a zero dimensional or 1 dimensional data in full format. As every data matrix in MATLAB is at least 2D, it becomes necessary to indicate its dimension for writing purposes.

7. ts
This is the text string that goes with the symbol. If nothing is entered then 'MATLAB data from GDXMRW' will be written in the GDX file.

See datalib example gdxmrw_wgdx01_init.

6.23.3 Extended use

In this section, we will discuss a few examples of the MATLAB and GAMS interface. We will give a simple example of a nonlinear optimization problem that would benefit from this capability and describe the steps that are needed in order to use our interface in this application.

- Special values
  Following example shows how special values are handled by this interface. It can be seen that rgdx can retrieve all these values from GDX file and display them appropriately in MATLAB.

```matlab
>> s.name = 'special';
>> s.form = 'full';
>> s.compress = true;
>> x = rgdx('sample', s)
```

```matlab
x =

    name: 'special'
```
Variables and Equations
In an optimization problem, we are not only interested in level value of variables and equations but also in their marginal values, lower and upper bounds. This interface gives its user ability to read any of these values into MATLAB. By default, rgdx and gams routines will read the level value of equations and variables but this can be changed very easily by using 'field' in input structure. In gams call user can also specify this in "$set matout" statement.

```matlab
$set matout ""matsol.gdx", x.m, dual.lo=dl ";
```
In this case the marginal value of variable 'x' will be read and lower bound of dual variable will be read and stored in 'dl'.

Text string and Text elements
GAMS allows its user to enter text string and explanatory text elements and all GDX file contain this information as well. Following example shows how to get these text elements in MATLAB.

```matlab
>> s1.name = 'el';
>> s1.te = true;
>> s1.ts = true;
>> s1.compress = true
s1 =
    name: 'el'
    te: 1
    ts: 1
    compress: 1
>> z = rgdx('sample', s1)
z =
    name: 'el'
    type: 'set'
    dim: 2
    val: [3x2 double]
    form: 'sparse'
    uels: {{1x2 cell} {1x2 cell}}
    ts: 'This is 2D set with text elements'
    te: {2x2 cell}
>> z.te
ans =
```
• String elements

One piece of information that may be needed within MATLAB is the modelstat and solvestat values generated by GAMS for the solves that it performed. This is easy to generate, and an example is given in do_status.m. This example is generated by taking the standard gamslib trnsport example, and adding the following lines to the end:

```matlab
$set matout "matsol.gdx", returnStat, str ";

Set
    stat / modelstat, solvestat /
    str / 'grunt', '%system.title%' /
    
Parameter returnStat(stat);
returnStat('modelstat') = transport.modelstat;
returnStat('solvestat') = transport.solvestat;
execute_unload %matout%;
```

Note that the relevant status numbers are stored in GAMS into the parameter returnStat which is then written to matsol.gdx and read back into MATLAB using the `rgdx` call.

```matlab
>> gamso.output = 'std';
>> gamso.compress = true;
>> gamso.form = 'full';
>> s = gams('trnsport')
```

```matlab
s =

1
1
```

See datalib example gdxmrw_ext01_init.

• Advanced Use: Plotting

One of the key features of the GAMS/MATLAB interface is the ability to visualize optimization results obtained via GAMS within MATLAB.

Some simple examples are contained with the program distribution. For example, a simple two-dimensional plot with four lines can be carried out as follows. First create the data in GAMS and export it to MATLAB using the `gams` routine (see APPENDIX C - Calling GAMS model from MATLAB for a detailed description of this routine).

We make an assumption that the user will write the plotting routines in the MATLAB environment. To create the plot in MATLAB, the following sequence of MATLAB commands should be input (saved as do_plot.m).

```matlab
gamso.output = 'std';
gamso.compress = true;
gamso.form = 'full';
[a,xlabels,legendset,titlestr] = gams('simple');
figure(1)
```
% Plot out the four lines contained in a;
% format using the third argument
plot(a,'+-');
% only put labels on x axis at 5 year intervals
xtick = 1:5:length(xlabels{1});
xlabels{1} = xlabels{1}(xtick);
set(gca,'XTick',xtick);
set(gca,'XTickLabel',xlabels{1});
% Add title, labels to axes
set(gca,'XTickLabel',xlabels{1});
xlabel('Year -- time step annual');
ylabel('Value');
% Add a legend, letting MATLAB choose positioning
legend(legendset{1},0);
% match axes to data, add grid lines to plot
axis tight
grid

The data is created using the following gams code.

$title Examples for plotting routines via MATLAB
$set matout "'matsol.gdx', a, t, j, sys ";

Set
   sys / '%system.title%' /
   t / 1990*2030 /
   j / a, b, c, d /
;
Parameter a(t,j);
a("1990",j) = 1;
loop(t, a(t+1,j) = a(t,j)*(1 + 0.04*uniform(0.2,1.8)););

Parameter year(*);
year(t) = 1989 + ord(t);

* Omit some data in the middle of the graph:
a(t,j)$((year(t) > 1995)*(year(t) <= 2002)) = na;
execute_unload %matout%

The following figure is an example created using this utility (and the MATLAB command print -djpeg simple).

See datalib example GDXMLRWPPlotting01.

MATLAB supports extensive hard copy output or formats to transfer data to another application. For example, the clipboard can be used to transfer meta files in the PC environment, or encapsulated postscript files can be generated. The help print command in MATLAB details the possibilities on the current computing platform.

Scaling of pictures is also most effectively carried out in the MATLAB environment. The following code is an example of rescaling printed out. Note that the output of this routine is saved as a jpeg file "rescale.jpg".
do_plt;
fpunits = get(gcf,'PaperUnits');
set(gcf,'PaperUnits','inches');
figpos = get(gcf,'Position');
pappos = get(gcf,'PaperPosition');
newpappos(1) = 0.25;
newpappos(2) = 0.25;
newpappos(3) = 4.0;
% get the aspect ratio the same on the print out
newpappos(4) = newpappos(3)*figpos(4)/figpos(3);
set(gcf,'PaperPosition',newpappos),
print -djpeg100 rescale.jpg
set(gcf,'PaperUnits',figpos);
set(gcf,'PaperUnits',fpunits);

Other examples of uses of the utility outlined in this paper can be found in the "m" files:
do_mpl
plotngon

that are contained in the distribution.

6.23.4 Acknowledgements

The authors would like to thank Alexander Meeraus and Michael R. Bussieck of GAMS corporation for constructive comments on the design and improvement of this tool. Thanks to Rishabh Jain for work on an earlier version.

6.23.5 APPENDIX A - Configuring GDXMRW

6.23.5.1 Installation

This section describes the installation procedure for all types of machines. The following section describes the testing procedure for verifying a correct installation.

First of all, you need to install both MATLAB and GAMS on your machine. For brevity, we will assume that the GAMS system (installation) directory is (for Windows)

C:\GAMS

and for non-Windows systems:

/usr/local/gams

All of the utilities come as a part of the GAMS distribution, so to use them you have only to add the GAMS directory to the MATLAB path. One way to do this is from the MATLAB command prompt, as follows:

>> addpath 'C:\GAMS'; savepath;

OR this can be done by following these steps:

1. Start MATLAB
2. Click on 'File' tab.
3. Now click on 'Set Path'
4. Click on 'Add Folder'
5. Select GAMS directory and click 'OK'.
6. Save it and then close it.
6.23.5.2 Testing

The GAMS system comes with some tests that you should run to verify the correct configuration and operation of the GDXMRW utilities. In addition, these tests create a log file that can be useful when things don’t work as expected. To run the tests, carry out the following steps.

1. Create a directory to run the tests in, e.g.
   ```
   mkdir /tmp
   ```

2. Extract the test models and supporting files from the GAMS test library into the test directory.
   ```
   cd /tmp
   testlib gdxmrw03
   testlib gdxmrw04
   testlib gdxmrw05
   testlib gdxmrw06
   testlib gdxmrw07
   ```

3. Execute the GAMS files `gdxmrw03`, `gdxmrw04`, and `gdxmrw06`. The files `gdxmrw03` and `gdxmrw04` test that the `rgdx` and `wgdx` routines are working properly, and `gdxmrw06` test `irgdx` and `iwgdx`. In addition to calling MATLAB in batch mode, they verify that the data are read and written as expected and give a clear indication of success or failure.

4. The GAMS file `gdxmrw05` tests the `gams` MATLAB routine. Like the other tests, it can be run in batch mode. You can also run it interactively by starting MATLAB, making `tmp` the current directory, and running the script `testinst.m`.
   ```
   >> testinst
   ```
   In addition to messages indicating success or failure, this test produces a log file `testinstlog.txt` that will be useful in troubleshooting a failed test.

5. The GAMS file `gdxmrw07` helps in basic testing if you find that GDXMRW is not functioning at all. To use this, open MATLAB, make `tmp` the current directory and run `gdxmrwSupport`. You can then look over the outputs and identify if and where any errors or failures are occurring. Re-run the commands that result in failures via cut-and-paste, and examine the tests and the output carefully: perhaps a solution will present itself to you. If the issue persists, send the screen output and the resulting log file `gdxmrwSupport.txt` (located in the created folder `tmp_gdxmrw`) to GAMS Support (support@gams.com).

6.23.6 APPENDIX B - Utility functions: gdxWhos and gdxInfo

In this section, we’ll describe two utilities, `gdxWhos` and `gdxInfo`, provided with GDXMRW that can be called from the MATLAB command prompt. The `gdxWhos` function is patterned loosely after the MATLAB function `whos` used to query `.mat` files. It provides information about symbols in a specified GDX file. The only input argument is the name of a GDX file, which can be with or without the ‘.gdx’ extension. With no output argument, `gdxWhos` lists the symbols in the specified GDX file in the MATLAB command prompt. When used with an output argument, information about the symbols in the GDX file is returned as an array of structures. This meta-data can be used in a variety of ways and is especially useful when programming.

An example GDXWhos call without an output argument is shown below:
The above output indicates that there are five indexed parameters in the file 'idx1_.gdx'. The way we can recognize an indexed parameter is by noticing numbered arguments of the parameters under the 'Name' column. For non-indexed data, the arguments will appear either as the name of a set or by a '∗'. For example, the first three symbols are non-indexed and the last symbol is indexed in the output from this call:

```
>> gdxWhos('fake.gdx');
Symbol info of GDX fake.gdx
   Index Type Dim NRecs Name
   1   Set     1   3   i(∗)
   2  Parameter 1     3   a(i)
   3     Set     1     3   d_i_m__3(∗)
   4  Parameter 1     3   aa(3)
```

The GDXInfo function is patterned after the gdxdump utility in GAMS. It lists and dumps all of the data values for each symbol in the specified GDX file to the MATLAB console. The only input argument is the name of a GDX file, which can be with or without the .gdx extension. It accepts no output arguments. It is especially useful to see the entire contents of smaller GDX files. To use the function, we could simply enter a command such as gdxInfo('idx1_') in the MATLAB command prompt, and the entire data contents would be displayed in the console. Further information about what is displayed can be found in the GAMS documentation of the gdxdump utility.

### 6.23.7 APPENDIX C - Calling GAMS model from MATLAB

Until now we have discussed the data Import/Export utility between MATLAB and GAMS. In this section, we will discuss a new MATLAB utility 'gams' that initializes a GAMS model with MATLAB data then executes GAMS on that model and bring the results back into MATLAB. This 'gams' routine is based on the same design as rgdx and wgdx but instead it does everything in one call. This routine can take multiple input arguments and can return multiple output arguments. Its standard syntax is as follows:

```
>> [x1, x2, x3] = gams('model', s1, s2.., c1, c2..);
```

Here note that the first argument of gams is the GAMS model name plus any user specific command line settings. If a user wants to solve the given model (in this case found in qp.gms) using a different solver then it can be done by adding that solver to the GAMS model name as "qp nlp=baron". This feature allows a user to change the execution time behavior of the model.

The rest of the input arguments of GAMS are structures. Their positioning is not important. These structures are of two kinds; one is similar to the input structure of wgdx and the other structure has just two string fields, name and val. This latter structure is used to set or overwrite values in the model using the "$set" variables syntax of GAMS. We will explain it in detail a later section.

The first step is to generate a working GAMS model. For example, we can set up a simple model file to solve a quadratic program that minimizes $\frac{1}{2} x^T Q x + c^T x$ subject to $Ax \geq b$ and $x \geq 0$.

The GAMS model for this quadratic problem is as given in the following code.
$set matout "'matsol.gdx', x, dual, obj, returnStat ";

Set
  i / 1*2 /
  j / 1*3 /;

Alias (j1,j);

Parameter
  Q(j,j1) / 1 .1 1.0
     2 .2 1.0
     3 .3 1.0 /
  A(i,j) / 1 .1 1.0
     1 .2 1.0
     1 .3 1.0
     2 .1 -1.0
     2 .3 1.0 /
  b(i) / 1 1.0
     2 1.0 /
  c(j) / 1 2.0 /

Variable obj;

Positive Variable x(j);

Equation cost, dual(i);

cost..   obj =e= 0.5*sum(j, x(j)*sum(j1, Q(j,j1)*x(j1))) + sum(j, c(j)*x(j));
dual(i).. sum(j, A(i,j)*x(j)) =g= b(i);

Model qp / cost, dual /;

$if exist matdata.gms $include matdata.gms

solve qp using nlp minimizing obj;

Set stat / modelStat, solveStat /;

Parameter returnStat(stat);
returnStat('modelStat') = qp.modelstat;
returnStat('solveStat') = qp.solvestat;

execute_unload %matout%;

This GAMS qp model can be executed directly at the command prompt using the following command

  gams qp (for Unix/Linux)

or

  gams.exe qp (for Windows)

or the user can simply hit the run button in the GAMSIDE. The optimal value is 0.5. In order to run the same model within MATLAB and return the solution vector x back into the MATLAB workspace, no change is required to the GAMS file. In MATLAB, all you have to do is to execute the following command:
This command will first collect the input structure data and create 'matdata.gdx' and 'matdata.gms' that contains include statements for the symbols written in a file matdata.gdx. In the previous example there is no structural input, so an empty 'matdata.gdx' file will be created and 'matdata.gms' will have just have a load statement for the GDX file but no load statements for any symbol. This is done to prevent any undesirable loading of data in the main model if there had already existed a 'matdata.gdx' or 'matdata.gms file'. After creating these two files then the 'gams' routine will execute "gams qp" using a system call. When this model is executed, another file 'matsol.gdx' will be created because of the execute_unload statement in the last line of the model. Here it should be noted that any model that you want to execute using the MATLAB gams routine should contain something like

```matlab
$set matout 'matsol.gdx', x, dual, obj, returnStat ;
```

either as the first line, or somewhere near the start of the model file. This is a standard gams $set statement, setting the value of the local variable 'matout'. The reason to have this statement near the start of the gms file is that the gams routine searches the file from the beginning for "$set matout" in the gms file. As these files can be very large, it is wise to have this statement near the start of the file. In this statement 'fileName' is the gdx file name that will be created containing symbols 'x1', 'x2', etc. These symbols can then be exported to MATLAB. The last line of the model should always be

```matlab
execute_unload %matout%;
```

The purpose of setting the first and last line of the model in this manner is to specify what data the user wants to export to MATLAB in a "header" of the model. As MATLAB does not give any information about the output arguments except the number of expected arguments, we have to specify what data to export to MATLAB in the GAMS model with minimum modification to the existing model. In the previous example, there is only one output argument, thus the gams routine will get data for its first element from the output gdx file and store it in the MATLAB output argument.

If there is more than one output argument, e.g.,

```matlab
>> [x, u] = gams('qp');
```

then the gams routine will read the output gdx file and store its first element information of the GDX file as the first output argument of MATLAB, i.e. 'x,' and the second element information of the GDX file in the second output argument of MATLAB, i.e. 'u' and so on. If the number of MATLAB output arguments is greater than the number of elements in the GDX file then gams will throw an error.

See datalib example gdxmrw_qp4.

### 6.23.7.1 Input Structure

As mentioned earlier, the gams routine takes input arguments in structured form. It allows two different types of structure input. One contains the symbol data similar to the wgdx input structure, to be exported to the GDX file. The other structure will just have two string fields 'name' and 'value'. Example use:

```matlab
>> s.name = 'Q';
>> s.val = eye(3);
>> s.form = 'full';
>> m = struct('name','m','val','2');
>> [x] = gams('qpmcp',s, m);
```
In this example both 's' and 'm' are structures but 'm' has only two fields and both are strings. The gams routine will use the 's' structure to create a 'matdata.gdx' file and 'm' to modify the execution command line to include "--m=2" at the end i.e. a command that will executed will be "gams qpmcp --m=2".

The structure 's' is the same as the input structure for wgdx but with two important differences. Firstly, it can be seen in the above example that 's' doesn't have a 'type' field. In wgdx we assume the type to be 'set' by default, but in the gams routine the type is assumed to be 'parameter' by default. The second change is an optional additional input field (in addition to those given in Section Input Structure) called "load".

- load
  It is a string input representing how the corresponding data will be loaded into the GAMS program. Depending on the value of the global option "gamso.input" (see next section) the input data will be read into GAMS in different ways. Suppose the input structure 's' has a 'name' field called 'foo'. By default (where gamso.input = 'compile'), the file matdata.gms will

$loadR foo

The GAMS parameter (or set) foo will be replaced by the data that is in the 'matdata.gdx' container called 'foo'. If the data has been initialized before in the model, this will replace that initial data with the new data from 'matdata.gdx'. The option can also be explicitly set using

s.load = 'replace'

There are two other compile time load options, namely 'initialize' and 'merge'. The first is only valid if the parameter values have not been initialized in the GAMS file, otherwise an error is thrown. It uses the GAMS syntax

$load foo

The merge option is valid when the GAMS file being run has already initialized the parameter values. The new values in the MATLAB structure 's' are merged into the parameter simply overwriting existing values with the new values given. Explicitly, the 'matdata.gms' file contains the statement

$loadM foo

to direct GAMS accordingly.

Finally, if gamso.input = 'exec', the loading will occur at execution time. In this case, s.load = 'initialize' is not a valid input, the default setting is s.load = 'replace' which carries out

execute_load "matdata.gdx" foo

and the alternative setting s.load = 'merge' carries out

execute_loadpoint "matdata.gdx" foo

In this way, the data is loaded at execution time and performs an appropriate replace or merge.

6.23.7.2 Global input to change default behavior

Until now we have seen how to specify different input to the gams routine and in this section we will see how to change the default behavior of a gams call. This can be done by creating a structure 'gamso' in the current workspace and adding different fields to that structure. There are currently nine fields that can be set in that structure that affect the behavior of the program. Except the uels field, all other string fields take case insensitive data. These are as follows:
• gamso.output
By default, output of the gams routine will be in structure form but it might be the case that a user is only interested in the data matrix, i.e., the val field of that structure. This can be done by setting gamso.output as 'std'. This will give only the value matrix as output. If this is not set to 'std' then output will be in the structure form described in the wgdx section.

```matlab
>> gamso.output = 'Std';
>> x = gams('qp nlp=baron')
```

```
x =
0.5000
```

• gamso.input
By default, the interface updates data at compile time. Thus, if execution time updates are made to the parameters before the line `'$include matdata.gms'` these may override the data that is provided in `matdata.gms` (i.e. from the command line). This may not be desirable. If you wish to perform execution time updates to the data, you should set gamso.input to 'exec'.

• gamso.write_data
If this is set to 'no', then all parameters on the call to gams are ignored, except the program name. This is useful for dealing with large datasets. Consider the following invocation:

```matlab
x = gams('largedata','A');
y = gams('resolve','A');
```

The first call generates a file `matdata.gms` containing the elements of the matrix A for use in the largedata.gms program. The second call rewrites a new `matdata.gms` file that again contains A. If we wish to save writing out A the second time we can use the following invocation:

```matlab
x = gams('largedata','A');
gamso.write_data = 'no';
y = gams('resolve','A');
clear gamso;
```
or the equivalent invocation:

```matlab
x = gams('largedata','A');
gamso.write_data = 'no';
y = gams('resolve');
clear gamso;
```

• gamso.show
This is only relevant on a Windows platform. This controls how the 'command box' that runs GAMS appears on the desktop. The three possible values are:

- 'minimized' (default): The command prompt appears iconified on the taskbar.
- 'invisible': No command prompt is seen.
- 'normal': The command prompt appears on the desktop and focus is shifted to this box.

• gamso.path
This option is used to specify fully qualified path for the gams executable. This is very useful if you have multiple versions of GAMS installed on your system and want to make sure which version you are running for the gams call. e.g.

```matlab
>> gamso.path = 'C:\Program Files\GAMS23.4\GAMS.exe';
```

The output of gams is similar to rgdx but unlike the rgdx gams routine it doesn't take input specific to a particular symbol. Thus it becomes important to implement a way to change the default behavior of the output. This can be achieved by adding following field to the global structure 'gamso'. All these fields behave similar to that described in rgdx and take the same input as of rgdx.
• gamso.compress
• gamso.form
• gamso.uels
• gamso.field

This is a global option however.

6.24  GDXRANK

A GAMS Utility for Ranking One-Dimensional Numeric Data.

Author

Thomas F. Rutherford, Department of Economics, University of Colorado
Paul van der Eijk, GAMS Development Corporation

Date

July, 2003

Source of this document: http://www.mpsge.org/gdxrank/index.html

This utility consists of an executable file, gdxrank.exe, and a GAMS libinclude file, rank.gms. The former resides in the GAMS system directory, and the latter resides in the inclib/ subdirectory of GAMS system.

6.24.1  Usage

6.24.1.1  Usage for GDXRANK

GDXRANK reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.

Usage:

gdxrank inputfile outputfile

Each one dimensional parameter is read from the input file, sorted and the corresponding integer permutation index written to the output file using the same name for the symbol. GAMS special values such as Eps, +Inf and -Inf are recognized.

6.24.1.2  Usage for RANK

$libInclude rank v s r [p]

The first three arguments are required. The last is optional. These are defined as following:
6.24 GDXRANK

<table>
<thead>
<tr>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>( v(s) )</td>
<td>Array of values to be ranked.</td>
</tr>
<tr>
<td></td>
<td>( s )</td>
<td>A one-dimensional set, the domain of array ( v ).</td>
</tr>
<tr>
<td>Output:</td>
<td>( r(s) )</td>
<td>Rank order of element ( v(s) ), an integer between 1 and ( \text{card}(s) ), ranking from smallest to largest.</td>
</tr>
<tr>
<td>Optional: (input and output)</td>
<td>( p(*) )</td>
<td>On input this vector specifies percentile levels to be computed. On output, it returns the linearly interpolated percentiles.</td>
</tr>
</tbody>
</table>

6.24.2 General Comments

- RANK only works for numeric data. You cannot sort sets.
- The first invocation must be outside of a loop or if block. This routine may be used within a loop or if block only if it is first initialized with blank invocations ("$libInclude rank" in a context where set and parameter declarations are permitted (See Example 3).
- The following names are used within these routines and may not be used in the calling program:

  rank_tmp  rank_u  rank_p

- This routine returns rank values and does not return sorted vectors, however rank values are easily used to produce a sorted array. This can be done using computed "leads" and "lags" in GAMS' ordered set syntax, as illustrated in examples 1 and 3 below.

6.24.3 GDXRANK Example

In this example we sort a parameter, create a sorted version and verify that the sort worked correctly:

```gams
Set I / i1*i6 /;
Parameter A(I) / i1 +Inf, i2 -Inf, i3 Eps, i4 10, i5 30, i6 20 /;
display A;
* write symbol A to gdx file
execute_unload "rank_in.gdx", A;
* sort symbol; permutation index will be named A also
execute 'gdxrank rank_in.gdx rank_out.gdx';
* load the permutation index
Parameter AIIndex(i);
execute_load "rank_out.gdx", AIIndex=A;
display AIIndex;
* create a sorted version
Parameter ASorted(i);
ASorted(i + (AIIndex(i) - ord(i))) = A(i);
display ASorted;
```
* check that the result is sorted
Set C(i);
C(i)=yes$(ord(i) < card(i)) and (ASorted(i) > ASorted(i+1));
display C;
abort$(card(C) <> 0) 'sort failed';

6.24.4 RANK Example

6.24.4.1 Example 1: Rank a vector, and display the data in sorted order

Set
i 'Set on which random data are defined' / a, b, d, c, e, f /
k 'Ordered set for displaying sorted data' / 1*6 /;
Parameter
x(i) 'Random data to be sorted'
r(i) 'Rank values'
s(k,i) 'Sorted data';
x(i) = uniform(0,1);
$libInclude rank x i r
display x;

* Generate a sorted list using the ordered set k.

* This assignment statement illustrates how the rank orders
* can be used to sort output for display in proper order. This
* statement uses GAMS support for computed "leads" and "lags"
* on the ordered set k. The loop is used to improve execution
* speed for larger dimensional sets:

loop(k$sameas(k,"1"),
s(k+(r(i)-1),i) = x(i);
);

option s:3:0:1;
display s;

This example is also part of the GAMS Data Utilities Library, see model [rank01] for reference. It writes
the following lines to ex1.lst:

---- 11 PARAMETER x Random data to be sorted
a 0.172, b 0.843, d 0.550, c 0.301, e 0.292, f 0.224

---- 75 PARAMETER s Sorted data
1.a 0.172
2.f 0.224
3.e 0.292
4.c 0.301
5.d 0.550
6.b 0.843
The RANK libinclude can be used to also sort variable levels as the following example shows:

Set  
  i 'Set on which random data are defined' / a, b, d, c, e, f /
  k 'Ordered set for displaying sorted data' / 1*6 /;

Variable  
  x(i) 'Random data to be sorted'

Parameter  
  r(i) 'Rank values'
  s(k,i) 'Sorted data';

x.l(i) = uniform(0,1);

$onDotL
$libInclude rank x i r
display x.l;
loop(k$sameas(k,"1"),
  s(k+(r(i)-1),i) = x(i);
);
option s:3:0:1;
display s;

Before the "$libInclude rank" the dollar control option $onDotL needs to be activated.

6.24.4.2 Example 2: Generate percentiles for a random vector

Set  
  i 'Set on which random data are defined' / a, b, d, c, e /
  p 'Percentiles (all of them)' / 0*100 /;

Parameter x(i) 'Random data to be sorted';

* Generate the random data on set i:
x(i) = uniform(0,1);
display x;

Parameter  
  r(i) 'Rank values'
  pct(*) 'Percentiles to be computed' / 20 20.0, median 50.0, 75 75.0 /;

* Generate ranks and compute the specified percentiles (Note that
* the rank array, r, is required, even if the values are not used.)

$libInclude rank x i r pct

* Display three percentiles:
display pct;

This example is also part of the GAMS Data Utilities Library, see model [rank02] for reference. The random data are displayed as follows in the listing file:
--- 11 PARAMETER x Random data to be sorted

a 0.172, b 0.843, d 0.550, c 0.301, e 0.292

The interpolated percentiles are computed as follows:

--- 103 PARAMETER pct Percentiles

20 0.268, 75 0.550, median 0.301

The following code evaluates a full set of percentiles, from 1 to 100. The GAMS special value of EPS is used to represent zero in the percentile calculation. (Percentiles between zero and one are not permitted to avoid misunderstandings about how percentiles are scaled.) The code makes use of Tom Rutherfords plot libInclude available at http://www.mpsge.org/gnuplot/index.html

\[
pct(p) = (\text{ord}(p) - 1) + \text{eps};
\]
\[
pct(\text{"median"}) = 0;
\]
display pct;

$\text{libInclude} \ rank \ x \ i \ r \ pct$
display pct;

* Plot the results using GNUPLOT:

\[\text{Set pl}(p) / 20, 40, 60, 80, 100 /;\]
\[\$\text{setGlobal} \ domain \ p\]
\[\$\text{setGlobal} \ labels \ pl\]
\[\$\text{libInclude} \ plot \ pct\]

This is the generated plot:

6.24.4.3 Example 3: Use GDXRANK to report multisectoral Monte Carlo results

One of the most perplexing challenges in economic modeling with GAMS is to present multisectoral results in an easily interpreted format. One simple idea is to present sectoral results in a sorted sequence to make it easier to identify the most seriously affected sectors. The presentation of results in a multisectoral model is made even more challenging when model results are generated for a randomized set of scenarios. A summary of Monte Carlo results involves reporting both mean results and their sensitivity. One means of characterizing the sensitivity of model results is to report functions of the sample distribution such as the upper and lower quartiles.

In this example, I illustrate how gdxrank can be used to help report results from the Monte Carlo analysis of a multisectoral model.

The complete model used as example here is part of the GAMS Data Utilities Library, see model [rank03] for reference. The following extract of that model generates the result we want to look at.
* Determine ranking of sectors by mean impact:
  \[
  \text{mean}(i) = qvalue(i,\text{"mean"});
  \]
  $\text{libInclude rank mean i meanrank}$

* The following statement creates a tuple matching the ordered
* set, \( ki \), to the set of sectors, \( i \). In this tuple, the sequence of
* assignments corresponds to increasing mean impacts:
  \[
  \text{imap}(ki+(\text{meanrank}(i)-\text{ord}(ki)),i) = \text{yes};
  \]

* Evaluate quartiles of sectoral impacts for each sector:
  \[
  \text{loop}(i, \\
  \text{x(s)} = v(s,i);
  \]
  
* Load quartile with the perctiles to be evaluated (25th, 50th and 75th):
  \[
  \text{quartile(qtl)} = qv(qtl);
  \]
  $\text{libInclude rank x s r quartile}$

* Save the quartile values:
  \[
  \text{qvalue}(i,qtl) = \text{quartile(qtl)};
  \]

\[
\text{display qvalue;}
\]

Parameter results(ki,i,*) 'Summary of impacts (sorted)';
results(ki,i,"mean")$\text{imap}(ki,i) = \text{mean}(i);$results(ki,i,qtl)$\text{imap}(ki,i) = \text{qvalue}(i,qtl);$display results;

The program produces the following display output:

<table>
<thead>
<tr>
<th></th>
<th>q25</th>
<th>q50</th>
<th>q75</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FOO</td>
<td>-13.776</td>
<td>-12.110</td>
<td>-10.671</td>
</tr>
<tr>
<td>2</td>
<td>MWO</td>
<td>-13.496</td>
<td>-11.982</td>
<td>-10.512</td>
</tr>
<tr>
<td>3</td>
<td>SCS</td>
<td>-12.244</td>
<td>-10.418</td>
<td>-8.738</td>
</tr>
<tr>
<td>5</td>
<td>ADM</td>
<td>-7.865</td>
<td>-5.601</td>
<td>-3.470</td>
</tr>
<tr>
<td>6</td>
<td>CNN</td>
<td>-6.437</td>
<td>-5.320</td>
<td>-4.404</td>
</tr>
<tr>
<td>7</td>
<td>0TH</td>
<td>-5.732</td>
<td>-4.781</td>
<td>-4.012</td>
</tr>
<tr>
<td>8</td>
<td>PIP</td>
<td>-3.142</td>
<td>-1.930</td>
<td>-0.836</td>
</tr>
<tr>
<td>9</td>
<td>0IN</td>
<td>-2.124</td>
<td>-1.339</td>
<td>-0.563</td>
</tr>
<tr>
<td>10</td>
<td>TPP</td>
<td>-2.527</td>
<td>-0.988</td>
<td>0.521</td>
</tr>
<tr>
<td>11</td>
<td>GEO</td>
<td>-1.542</td>
<td>-0.826</td>
<td>-0.196</td>
</tr>
<tr>
<td>12</td>
<td>AGF</td>
<td>-1.178</td>
<td>-0.624</td>
<td>-0.072</td>
</tr>
<tr>
<td>13</td>
<td>ECM</td>
<td>0.137</td>
<td>0.466</td>
<td>0.796</td>
</tr>
<tr>
<td>14</td>
<td>SSM</td>
<td>0.303</td>
<td>0.711</td>
<td>1.079</td>
</tr>
<tr>
<td>15</td>
<td>CON</td>
<td>0.604</td>
<td>1.071</td>
<td>1.479</td>
</tr>
<tr>
<td>16</td>
<td>PST</td>
<td>0.814</td>
<td>1.924</td>
<td>2.966</td>
</tr>
<tr>
<td>17</td>
<td>RLW</td>
<td>1.885</td>
<td>2.787</td>
<td>3.632</td>
</tr>
<tr>
<td>18</td>
<td>0FU</td>
<td>2.789</td>
<td>3.300</td>
<td>3.829</td>
</tr>
<tr>
<td>19</td>
<td>0LE</td>
<td>3.155</td>
<td>3.563</td>
<td>3.964</td>
</tr>
<tr>
<td>20</td>
<td>ELE</td>
<td>3.480</td>
<td>3.896</td>
<td>4.318</td>
</tr>
<tr>
<td>21</td>
<td>PSM</td>
<td>3.733</td>
<td>4.134</td>
<td>4.591</td>
</tr>
<tr>
<td>22</td>
<td>CAT</td>
<td>3.216</td>
<td>4.417</td>
<td>5.461</td>
</tr>
<tr>
<td>23</td>
<td>TRO</td>
<td>3.953</td>
<td>5.123</td>
<td>6.549</td>
</tr>
<tr>
<td>24</td>
<td>OLP</td>
<td>4.614</td>
<td>5.262</td>
<td>5.909</td>
</tr>
</tbody>
</table>
The tabular report is helpful, but it does not convey the results as immediately as a picture. GNUPLOT's errorbar plot format is a convenient graphical format for portraying this information. The libinclude interface to GNUPLOT does not support this type of plot, so the continuation of the program produces the GNUPLOT command and data files before invoking the GNUPLOT program:

* Write out a GNUPLOT file to generate a chart of the results:

```plaintext
File kplt / ex3.gnu /;
put kplt;
kplt.lw = 0;

put "reset"/;
put 'set title "Sectoral Impacts with Quartiles"'/;
put "set linestyle 1 lt 8 lw 1 pt 8 ps 0.5"/;
put "set grid"/;
put 'set ylabel "% change"'/;
put "set xzeroaxis"/
put "set xlabel 'sector'"/
put 'set xrange [1:',card(i),']'/
put 'set xtics rotate ('
loop(ki, loop(i$imap(ki,i), put ord(ki):0,qvalue(i,"mean"), qvalue(i,"q25"), qvalue(i,"q75")/;));
put @(file.cc-1) ')'/
put "plot 'ex3.dat' notitle with errorbars ls 1"/;
putClose;

File kpltd data / ex3.dat /;
put kpltd data;
kpltd data nr = 2;
kpltd data nw = 14;
kpltd data nd = 6;
loop(ki, loop(i$imap(ki,i), put ord(ki):0,qvalue(i,"mean"), qvalue(i,"q25"), qvalue(i,"q75")/;));
putClose;
execute 'wgnupl32 ex3.gnu -';
```

This is the generated plot:

6.24.4.4 Example 4: Repeated computation of percentiles within a loop

The following code extract shows how to do repeated computation of percentiles within a loop while the data is changing. This is an extract of the model [rank04] from the GAMS Data Utilities Library.
Do several iterations, computing percentiles in each step:

```plaintext
loop(iter,
  * Substitute a call to the NLP solver by a call to the random
  * number generator. In many applications, this substitution
  * produces profoundly more sensible results.
  *
  * solve catchment using nlp maximizing max;
    z(week) = uniform(0,1);
  *
  * If you want to retrieve percentile values, you need to reassign
  * the percentiles that you wish to retrieve at this point in the
  * program. If pct() were not reassigned at this point, the INPUT
  * values would correspond to the OUTPUTs from the previous call.
    pct(pctl) = pct0(pctl);
  *
$ libInclude rank z week rnk pct
  pctval(iter,pctl) = pct(pctl);
  );
  display pctval;
```

Output:

```plaintext
58 Here are the INPUT values of PCT0 and PCT prior to the call to rank:
50 50.000,  75 75.000,  80 80.000,  95 95.000

58 Here are the values of PCT0 and PCT after the call to rank:
50 50.000,  75 75.000,  80 80.000,  95 95.000

Note that rank has changed the OUTPUT value of pct

50 0.424,  75 0.662,  80 0.712,  95 0.864

273 Percentile values in successive iterations
50 75 80 95
```
Example 5: Use GDXRANK generating percentiles for heterogenous households.

Set h / 0*100 /

Parameter
  y(h) 'Aggregate expenditure associated with household type h'
  n(h) 'Number of persons associated with household type h'
  ypc(h) 'Per-capita expenditure of household type h'
  rank(h) 'Rank of household in per-capita expenditure';

* Assign some random values:
  y(h) = uniform(0.2,1.2);
  n(h) = uniform(1,6);
  ypc(h) = y(h)/n(h);

* Assign ranks to household based on per-capita expenditures:
$libInclude rank ypc h rank

* Now determine percentile ranking of the households taking into account
  * differences in numbers of members and household representation:
Set r 'Temporary set used for ranking' / r0*r100 /

Parameter
  pcttmp(r) 'Temporary array for computing percentiles'
  pct(h) 'Percentile rankings for households';

Set r0(r) / r0 /;

* First, create an array with households assigned
loop((r0(r),h), pcttmp(r+(rank(h)-1)) = n(h););

  loop(r, pcttmp(r) = pcttmp(r) + pcttmp(r-1););
  pcttmp(r) = pcttmp(r)/sum(h, n(h));

loop((r0(r),h), pct(h) = pcttmp(r+(rank(h)-1)););

Parameter ranking 'Ranking of households and expenditures';

  loop((r0(r),h),
      ranking(r+(rank(h)-1),h,"n") = n(h);
      ranking(r+(rank(h)-1),h,"ypc") = ypc(h);
      ranking(r+(rank(h)-1),h,"pct") = pct(h);
  );

display ranking;
This example is also part of the GAMS Data Utilities Library, see model [rank05] for reference. It produces the following output:

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>ypc</th>
<th>pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>.43</td>
<td>5.662</td>
<td>0.044</td>
</tr>
<tr>
<td>r1</td>
<td>.8</td>
<td>5.682</td>
<td>0.047</td>
</tr>
<tr>
<td>r2</td>
<td>.58</td>
<td>4.477</td>
<td>0.052</td>
</tr>
<tr>
<td>r3</td>
<td>.14</td>
<td>5.742</td>
<td>0.058</td>
</tr>
<tr>
<td>r4</td>
<td>.55</td>
<td>4.815</td>
<td>0.063</td>
</tr>
<tr>
<td>r5</td>
<td>.65</td>
<td>3.702</td>
<td>0.063</td>
</tr>
<tr>
<td>r6</td>
<td>.61</td>
<td>5.880</td>
<td>0.064</td>
</tr>
<tr>
<td>r7</td>
<td>.54</td>
<td>4.463</td>
<td>0.064</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r98</td>
<td>.62</td>
<td>1.134</td>
<td>0.640</td>
</tr>
<tr>
<td>r99</td>
<td>.10</td>
<td>1.673</td>
<td>0.716</td>
</tr>
<tr>
<td>r100</td>
<td>.73</td>
<td>1.053</td>
<td>1.076</td>
</tr>
</tbody>
</table>

### 6.25 GDXRENAME

The GDXRENAME program renames the unique elements in a GDX file using the unique elements from a second GDX file.

#### 6.25.1 Usage

```
gdxrename data-file map-file {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

The GDXRENAME utility renames the unique elements in the data-file using a two dimensional set called map in the map-file. The renaming of the unique elements only affects the string stored for each element, and does not change the data order for the symbols in the data-file. Because no data is changing in the data-file, only the strings for the unique elements, these changes are applied to the data-file directly and no new data-file is created.

**data-file** = *fileName* (default = none)

Name of the file where the renaming is to be applied.

**map-file** = *fileName* (default = none)

Name of the file where the map set appears containing the renaming tuples.
6.25.2 Options

-**reverse** (disabled by default)

   Synonym: -r

   Reverse the direction of the rename operation; see below

6.25.3 Example

A simple example, where the labels of the parameter $A$ are renamed according to the map specified.

Set c / r, g, b, y /;
Parameter A(c);
A(c) = ord(c);
execute_unload 'data.gdx', A;
execute 'gdxdump data.gdx';
Set map(*,*) / r.red, g.green, b.blue, y.yellow /;
execute_unload 'map.gdx' map;
execute 'gdxrename data map';
Scalar rc;
rc = errorLevel;
abort$(rc <> 0) 'GDXRENAME execution error';
execute 'gdxdump data.gdx';

Symbol $A$ before the rename:

Parameter A(*) /
' r' 1,
' g' 2,
' b' 3,
' y' 4 /;

Symbol $A$ after the rename:

Parameter A(*) /
' red' 1,
' green' 2,
' blue' 3,
' yellow' 4 /;
6.26 GDXRRW

Interfacing GAMS and R

Author

Michael C. Ferris (ferris@cs.wisc.edu), Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706
Steven Dirkse (sdirkse@gams.com) GAMS Development Corp.

GDXRRW is a suite of utilities to import/export data between GAMS and R (both of which the user is assumed to have already) and to call GAMS conveniently from R. The software gives R users the ability to use all the optimization capabilities of GAMS, and allows visualization and other operations on GAMS data directly within R.

The GDXRRW tool is unique among the GDX interface utilities in that it is an R extension made available as an R package. As such, it is run as part of an R session or script, not as part of a GAMS run, and it follows the usual R package conventions.

1. **Installation:** The R software is designed to be easily extended. Thousands of extension packages are freely and conveniently available online and can be installed easily using a simple, standard procedure. The GDXRRW package is one of these. The latest version is available on GitHub in both source and binary form, along with a FAQ list, some hints and tips on common problems and solutions, and other helpful content. For convenience, the source and binary packages are also available in the gdxrrw directory of the GAMS distribution.

2. **Documentation:** The GDXRRW documentation is installed as part of the GDXRRW package and is available from within R in the usual way. See the GDXRRW GitHub page for details and hints.

3. **Examples, tests, and data:** Like many R packages, GDXRRW comes with examples and data that help a user get started using the package. The R help system and the GDXRRW GitHub provide pointers to these. GDXRRW GitHub page also provides hints on finding and running the thousands of lines of tests that come with the GDXRRW package.

6.27 GDXTROLL

gdxtroll reads TROLL data files into a GDX container.

6.27.1 Usage

gdxtroll in [out]

where

in

TROLL data text file (default extension is .txt)

out

GDX container file name (default in.gdx)

Accepts SPECS:NUMeríc and the following periodicities scalar NA, annual yyyyA, quartely yyyyQq, monthly yyyyMm weekly yyyyWwww (ISO 8601)
6.28 GDXVIEWER

6.28.1 Overview

GDXVIEWER is a tool to view and convert data contained in GDX files.

Besides inspecting a GDX file, gdxviewer allows you to export to a large number of data formats, including ASCII text, CSV, HTML, XML, database, and spreadsheet formats.

This tool is designed as an interactive Windows program, but it can also be operated through command line parameters.

6.28.2 Requirements

GDXVIEWER runs only on PC’s running Windows (95/98/NT/XP). The DLL GDXDCLIB.DLL needs to be in the same location as GDXVIEWER.EXE. If XLS files are saved, MS Excel needs to be present. If MDB database files are saved, MS Access needs to be present.

If GDXDCLIB.DLL is not found in the same directory as the executable gdxviewer.exe, the following window will be shown:

A simple way to make sure that GDXVIEWER has access to the GDXDCLIB.DLL dynamic load library is to place gdxviewer.exe in the GAMS system directory, e.g. c:\program files\GAMS21.3.

6.28.3 Creating GDX files

GDX files are binary data files. They can contain sets, parameters (including scalars), equations and variables. These files can be generated by a number of tools: by GAMS itself, by utilities such as MDB2GMS, SQL2GMS, GDXXRW.

To save all data from a GAMS model into a GDX file you can use the GDX=fln command line parameter:

```
C:\> gams trnsport.gms GDX=trnsport.gdx
```

From the IDE you can specify the command line parameter GDX=trnsport.gdx in the parameter edit box:
To selectively place identifiers in a GDX file you can use the `execute_unload` statement:

```gams
Model transport / all /;
solve transport using lp minimizing z;
display x.l, x.m;
execute_unload 'results.gdx', i, j, x;
```

In this example the sets `i` and `j` and the variable `x` are saved to the GDX file `results.gdx`.

Other ways to create GDX files include:

- The `MDB2GMS` tool can be used to convert data stored in MS Access tables to GDX files.
- The `SQL2GMS` tool can read data from virtually any SQL database (including any ODBC accessible database) and can create GDX files.
- The tool `GDXXRW` allows data from an Excel spreadsheet to be stored in a GDX file.
- `$GDXOUT` allows you to write data to a GDX file during GAMS compile time. This is not as useful as `execute_unload` but may have its use in special cases.
- You can write your own program to write a GDX file. There is an API and bindings for different languages such as Delphi, Kylix, VB6, VBA, VB.NET, C/C++, C#, Java, Fortran.
6.28.4 Viewing GDX files

After loading a GDX file in gdxviewer the content of the file is displayed in list view. The left-hand side of the window shows the index of the GDX file organized in a tree structure. When clicking on an identifier, the right-hand-side will display the actual data for the identifier.

When variables are shown, more information is available, such as bounds (lower and upper bounds) and marginals.

The GDX file can be loaded interactively using the File|Open menu, or it can be launched from the command line:

C:\> gdxviewer e:\models\trnsport.gdx

The command line specification can also be used to launch gdxviewer from within a GAMS model as in:

Model transport / all /;
solve transport using lp minimizing z;
display x.l, x.m;
execute_unload 'results.gdx', i, j, x;
execute '=gdxviewer results.gdx';

In this case gdxviewer.exe was located in the GAMS system directory, such that execute had no problems in finding it.

Note: there are alternative tools to view GDX files. The GAMS IDE has a built-in GDX file viewer (use File|Open) and there is a command line utility called GDXDUMP.
6.28.5 Exporting an identifier

When the right mouse button is clicked on an identifier a pop-up menu is presented that allows you to export an identifier to a number of target formats.

The same operation can be invoked from the File|Export menu:
6.28.6 Exporting to a Text File

The text file export facility (File|Export|Text File) will write a GAMS identifier to a standard ASCII text file. Such a text file can look like:

seattle new-york 2.5
seattle chicago 1.7
seattle topeka 1.8
san-diego new-york 2.5
san-diego chicago 1.8
san-diego topeka 1.4

The separator symbol can be set using the menu Options|Configuration|Text File:

Other options that involve the format of the text file being written are: Options|Configuration|Export and Options|Configuration|Special Values. Currently there are no facilities to write fixed format text files. If you need to write fixed format text files you can use the GAMS PUT statement.
6.28.7 Exporting to a CSV files

Comma-separated Values (File|Export|CSV File) is a popular format to exchange data between applications. An example of such a file is:

'nyc', 325
'chicago', 300
'topeka', 275

Strings are surrounded by quotes and each field is separated by a comma. The precise format can be specified using the menu Options|Configuration|CSV File:

Other options that involve the format of the CSV file being written are: Options|Configuration|Export and Options|Configuration|Special Values.

6.28.8 Exporting to an XLS file

A GAMS identifier can be exported directly to an MS Excel spreadsheet using File|Export|Excel XLS File:
There are a few options available for this operation. Under Options|Configuration|Excel the following settings can be changed:

- Create Backup File
- Delete existing sheets

Excel can create a backup file. This file has a name of the form 'Backup of xxxxlk'. Deleting existing sheets will erase existing information in the spreadsheet. If not selected a new sheet will be added and old ones are retained.

Other options that involve the format of the CSV file being written are: Options|Configuration|Export and Options|Configuration|Special Values. Exporting to Excel is only available if you have Microsoft Excel installed on your machine.

Note: We can write all symbols in the gdx file to Excel by specifying ID=* on the command line.
6.28.9 Exporting to an XLS Pivot Table

We can export to an XLS file and create a Pivot Table automatically (File|Export|Excel Pivot Table):

Pivot tables are a very convenient way to analyze multi-dimensional data.

The following options are available: Options|Configuration|Excel, Options|Configuration|Export and Options|Configuration|Special Values.

6.28.10 Exporting to a GAMS Include Files

The option File|Export|GAMS Include file will export an identifier to a GAMS include file format. An example of such an exported include file can look like:

Parameter d 'distance in thousands of miles'
  / seattle.new-york 2.5, san-diego.new-york 2.5,
  seattle.chicago 1.7, san-diego.chicago 1.8,
  seattle.topeka 1.8, san-diego.topeka 1.4 /;
6.28.11 Exporting to an Access Tables

GDXVIEWER can export data directly to a table in an Access database using File|Export|Access (MDB or ACCDB) File. The name of the table will be the name of the parameter. If the table already exists, GDXVIEWER will try to create a new table with a slightly different name (e.g. d2, d3,...).

An option Options|Configuration|Access allows you to set the length of the text fields where the GAMS indices are stored. This length is used when creating the table.
A feature added in version 2.9 is the possibility to use intermediate CSV (comma separated value) files instead of using direct SQL INSERT statements. The CSV files can be read into Access using a bulk operation and is therefore faster for large datasets. When using CSV files make sure double quotes are used (if single quotes are used they will become part of the data). The temporary CSV files will be written to the Windows TEMP directory (e.g. C:\Windows\Temp). When the import is done, these scratch files will be removed automatically. If you want to look at the CSV files that are being fed into Access, export the data to a CSV file.

6.28.12 Exporting to an SQL Table

It is possible to export data to SQL databases through ADO which includes all databases accessible through ODBC. The configuration information can be specified in Options|Configuration|SQL Database.
The Test Connection button will allow you to check the configuration and see if the database can be connected to.

The SQL data for double precision number is no always the same for each database. E.g. for MS Access you can use double while for MS SQL server you can use float.

When exporting data a new table is created with the name of the identifier. If such a table already exists, names like name2, name3, are tried.

**6.28.13 Exporting to MS SQL Server**

We can export to Microsoft SQL Server through the standard SQL export facility. However a special facility called BULK INSERT is only available through the specialized SQL Server export tool. BULK INSERT writes a TAB delimited text file to the Windows TEMP directory and subsequently calls BULK INSERT to load that file. This way is often much faster that using individual INSERT statements for each record.

**6.28.14 Exporting to SQL Insert script**

An SQL script with INSERT statements like:

```sql

```
can be generated with File|Export|SQL Insert script. The following settings in Options|Configuration|SQL Insert were used:

<table>
<thead>
<tr>
<th>DB Columns</th>
<th>GDK column:</th>
<th>Quote:</th>
<th>Table name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>city1</td>
<td>dim1</td>
<td>✔</td>
<td>dist</td>
</tr>
<tr>
<td>city2</td>
<td>dim2</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>distance</td>
<td>level</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL INSERT statements will be generated of the form: INSERT INTO tablename(dbcoll,dbcoll2,...) VALUES (gdxcol1,gdxcol2,...). Column values can be quotes and INSERT statements are often terminated with a semicolon.
6.28.15 Exporting to SQL Update script

An SQL script with UPDATE statements like:

can be generated with File|Export|SQL Update script. The following settings in Options|Configuration|SQL Update were used:

<table>
<thead>
<tr>
<th>DB Columns</th>
<th>GDX column:</th>
<th>Quote:</th>
<th>Table name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>city1</td>
<td>dim1</td>
<td></td>
<td>dist</td>
</tr>
<tr>
<td>city2</td>
<td>dim2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>distance</td>
<td>level</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL UPDATE statements will be generated of the form: UPDATE tablename SET dbcol1=gdxval1 dbcol2=gdxval2 WHERE dbcol3=gdxdim1 and dbcol4=gdxdim2. Column values can be quotes and UPDATE statements are often terminated with a semicolon.

6.28.16 Exporting HTML

GDXVIEWER can write an identifier to an HTML file using File|Export|HTML File.
The options relevant to this format are specified in **Options**|**Configuration**|**HTML**.

![GDXViewer](image)

6.28.17 Exporting XML

GDXVIEWER can write an identifier to an XML file using **File**|**Export**|**XML File**.
The XML tags can be specified in Options|Configuration|XML.
6.28.18 Exporting fields

The menu Options|Configuration|Export allows you to set which fields are exported.

The following table gives the possibilities for exports:

<table>
<thead>
<tr>
<th></th>
<th>set</th>
<th>scalar</th>
<th>parameter</th>
<th>variable</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Lowerbound</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Level/Value</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Upperbound</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Marginal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

6.28.19 Special Values

GAMS data can assume so called special values: -INF, +INF, EPS, NA, and UNDF. The meaning of these special values is as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-INF</td>
<td>Minus infinity. Mostly used for non-binding lowerbounds.</td>
</tr>
<tr>
<td>+INF</td>
<td>Plus infinity. Mostly used for non-binding upperbounds.</td>
</tr>
<tr>
<td>EPS</td>
<td>Mostly used for marginals where it can indicate non-basic but numerically zero.</td>
</tr>
<tr>
<td>NA</td>
<td>Not available. Not often used.</td>
</tr>
<tr>
<td>UNDF</td>
<td>Undefined. Not often used.</td>
</tr>
</tbody>
</table>

When exporting GAMS identifiers we need to map such values to strings that the receiving program can understand. E.g. we could map -INF to -1.0e10 and +INF to +1.0e10. A good choice for EPS would be 0.0.

The mapping can be specified in Options|Configuration|Special Values:
When we export to a GAMS include file all special values are understood, so the mapping is not used. The defaults button will reset the mapping to their default values.

### 6.28.20 Plotting Data

GDXVIEWER has a built-in facility to quickly plot data. It includes LINE, BAR and PIE charts, examples are shown below. The plots can be made through the menu File|Plot.
For multi-dimensional data it may be needed to take a “slice” of the data to make meaningful graphs. In the example above we plotted a two dimensional quantity \( \text{vf} \) which looks like:
In this case we want to plot a pie graph of `vf(*,'mexico-df')` which can be specified in the index-selection tab:
6.28.21 Cube View

GDXVIEWER has a Cube View which allows to select rows and columns in a flexible way. In the example below we show a six dimensional variable where three dimensions are fixed, one dimension is chosen for the rows and two dimensions are chosen for the columns.

Below are some of the possibilities using parameter $d(i,j)$ from the trnsport.gms model:

6.28.22 Exporting cubes

After creating a cube view, we can export that configuration by a right mouse click:
Exporting a cube will only export the selected slice (if certain dimensions are held fixed) and depending on the target format it will preserve the layout, e.g. an exported aligned text file can look like:

```
new-york  chicago  topeka  
seattle    2.5   1.7   1.8  
san-diego  2.5   1.8   1.4  
```

Similarly, the XLS file can look like:

![XLS](image)

### 6.28.23 Commandline operation

The GDXViewer utility from version 2.3 accepts several command line parameters, so it can be used in a batch environment. When running in batch mode, the same configuration and option settings are used as for the interactive system and they can be changed by running GDXviewer interactively using the Options menu (the settings are saved in an INI file). It is advised to first run the program interactively until the results are as intended.

- **Single parameter** A single parameter is the filename of the GDX file. GDXViewer will load this file, and will continue to run interactively. Example: .

  `gdxviewer.exe test.gdx`

- **XLS writing** To write an XLS file, one can use the syntax `i=inputfile.gdx xls=outputfile.xls id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx xls=d:\tmp\result.xls id=x';
  ```

- **Text file writing** To write a text file, one can use the syntax `i=inputfile.gdx txt=outputfile.txt id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx txt=d:\tmp\result.txt id=x';
  ```

- **CSV file writing** To write a CSV file, one can use the syntax `i=inputfile.gdx csv=outputfile.csv id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx csv=d:\tmp\result.csv id=x';
  ```
• **HTML file writing** To write an HTML file, one can use the syntax `i=inputfile.gdx html=outputfile.html id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx html=d:\tmp\result.html id=x';
```

• **XML file writing** To write an XML file, one can use the syntax `i=inputfile.gdx xml=outputfile.xml id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx xml=d:\tmp\result.xml id=x';
```

• **GAMS include file writing** To write a GAMS include file, one can use the syntax `i=inputfile.gdx inc=outputfile.inc id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx inc=d:\tmp\result.inc id=x';
```

• **Access MDB file writing** To write a Access MDB file, one can use the syntax `i=inputfile.gdx mdb=outputfile.mdb id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx mdb=d:\tmp\result.mdb id=x';
```

• **Excel Pivot Table writing** To write a file containing a pivot table, one can use the syntax `i=inputfile.gdx pivot=outputfile.xls id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("'). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

• **SQL Database Table writing** To write a table to an SQL database, first interactively configure the connection to the database. The Export SQL Database option allows you to see if a connection succeeded and if the correct database was accessed. The configuration information is written to the SQLVIEWER.INI configuration file. The information in this file is used also when performing a batch command-line operation. The syntax is: `i=inputfile.gdx sql id=x`. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx sql id=x';
```

If you need to access several different databases, you can copy the file SQLVIEWER.INI (located in the directory where SQLVIEWER.EXE is placed). To tell GDXVIEWER to read a different INI file, you can say:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx ini=copy.ini sql id=x';
```
GDXVIEWER uses the MS Access and MS Excel applications as COM Object to write files in XLS (both XLS and PIVOT commands) or MDB format. Those applications may write to C:\My Documents in case no full path is specified. Other formats use the default GAMS working directory. In case when running under the IDE this is the location of the project file (*.GPR).

If a path or file name contains a blank, then it is possible to surround the name by double quotes as in:

```plaintext
execute_unload 'result.gdx',x;
execute 'gdxviewer.exe i=result.gdx csv="c:\my documents\result.csv" id=x';
```

Under windows 98 ME the call

```plaintext
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

will cause GAMS to continue while GDXVIEWER is executing. If we use:

```plaintext
execute '=gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

GAMS will wait until gdxviewer.exe is terminated before executing more statements. This situation is different under other operating systems such as XP and NT.

6.28.24 Notes

GDX

GDX stands for Gams Data Exchange. It is a binary file format to get data in and out of GAMS.

6.29 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet.

Note

GDXXRW is available on Windows only. This is an important factor to consider when moving an existing code to GAMS Engine.

6.29.1 Usage

gdxxrw inputFile {outputFile} {options} [symbols]

Options and symbol specifications can also be read from a text file; the use of an option file is indicated by preceding the file name with a @ (At sign.). When reading from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.

Options and symbol specifications can also be read from an area in a spreadsheet; see index below.

Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. The use of file names with embedded blanks is allowed as long as the file name is enclosed in double-quotes (").

Note

A small utility program is available to see if Excel is installed, to close an Excel file etc. See XLSTALK for more details. To read data from an Excel file without Excel installed see XLSHIMP.
### 6.29.2 Options

Describing the actions to be taken by GDXXRW requires passing a number of options and symbol specifications to the program. The ability of GDXXRW to process multiple actions in a single call makes the option passing and interpretation more complex.

There are four kinds of options:

1. **Immediate** Immediate options are recognized and processed before any other actions are taken and can only be specified once. Examples are:
   ```
   input= output= trace=
   ```

2. **Global** Global options are interpreted from left to right and affect every action that follows. The same option can be used multiple times to affect the actions that follow. Examples are:
   ```
   skipEmpty= epsOut=
   ```

3. **Symbol** A symbol definition introduces a new action for reading or writing a symbol. Examples are:
   ```
   par= set= dSet=
   ```

4. **Symbol attributes** Attributes specify additional information for the last symbol defined. Examples are:
   ```
   dim= cDim= merge clear
   ```

#### 6.29.2.1 Immediate Options

Immediate options are recognized independent of their position on the command line. They are global and they can only be specified once.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>none</td>
<td>Specify the input filename (required).</td>
</tr>
<tr>
<td>output</td>
<td>inputFileName</td>
<td>Specify the output filename.</td>
</tr>
<tr>
<td>log</td>
<td>none</td>
<td>Specify the log filename.</td>
</tr>
<tr>
<td>logAppend</td>
<td>none</td>
<td>Appending the log information to the file specified.</td>
</tr>
<tr>
<td>index</td>
<td>none</td>
<td>Indicates reading the options and symbols directly from the spreadsheet.</td>
</tr>
<tr>
<td>password</td>
<td>none</td>
<td>Password for an encrypted input file.</td>
</tr>
<tr>
<td>rWait</td>
<td>0</td>
<td>Delay after opening a spreadsheet.</td>
</tr>
<tr>
<td>checkDate</td>
<td>disabled</td>
<td>Write GDX file only if the input file is more recent than the GDX file.</td>
</tr>
<tr>
<td>useRC</td>
<td>disabled</td>
<td>Use Row-Column notation to specify cells and ranges.</td>
</tr>
<tr>
<td>reCalc</td>
<td>N</td>
<td>Controls if recalculations of cells inside Excel are executed after writing to the spreadsheet.</td>
</tr>
<tr>
<td>trace</td>
<td>1</td>
<td>Controls the amount of information written to the log.</td>
</tr>
<tr>
<td>maxDupeErrors</td>
<td>0</td>
<td>Maximum number of duplicate records allowed for a symbol.</td>
</tr>
<tr>
<td>updLinks</td>
<td>0</td>
<td>Updating of cells that refer to other spreadsheets.</td>
</tr>
<tr>
<td>runMarcos</td>
<td>0</td>
<td>Execution of Excel Auto macros.</td>
</tr>
</tbody>
</table>
Some more detailed remarks on the immediate options:

**input** = fileName (required, default = none)

    Synonym: i

    Either use the keywords input or i to specify the input file name anywhere on the command line or just specify the input file name without keyword at the first position behind GDXXRW. The file extension of the input file is required and determines the action taken by the program.

    The extension .gdx for the input file will read data from a GDX file and write data to a spreadsheet. The extension .xls, .xlsx, .xlsm, or .xlsb for the input file will read a spreadsheet and write the data to a .gdx file. In addition to the .xls, .xlsx, .xlsm, or .xlsb input file extension, the following file extensions are also valid for spreadsheet input files: .wk1, .wk2, .wk3 and .dbf.

    A file sharing conflict will arise when writing to a spreadsheet with the target file open in Excel. Either close the file in Excel before executing GDXXRW, or mark the spreadsheet as a shared workbook in Excel. To change the shared status of a workbook, use the Excel commands available under: Tools|Share Workbook.

    Writing to a shared workbook can be painfully slow; simply closing the file and reopen the file after GDXXRW has finished is often a better option.

**output** = fileName (default = inputFileName)

    Synonym: o

    When an output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. The file type, i.e. the file extension, depends on the installed version of Excel. Versions prior to Excel 2007 use the .xls file extension, later version use .xlsx, .xlsm, and .xlsb. Excel 2007 can write .xls files, but in that case the output file has to be specified with an .xls file extension.

**log** = fileName (default = none)

    Specifies the filename of the log file. When omitted, log information will be written to standard output. When using GDXXRW in a GAMS model that is started from the GAMS IDE, the output will be written to the IDE process window.

**logAppend** = fileName (default = none)

    Using logAppend will add the log information to the end of the file specified. If the file does not exist yet, a new one will be created.

**index** = Excel Range

    The index option is used to obtain the global options, symbols and symbol attributes specified by reading them from the spreadsheet directly. The parameters are read using the specified range, and treated as if they were specified directly on the command line. The first three columns of the range have a fixed interpretation: dataType, Symbol identifier and dataRange. The fourth and following columns can be used for additional parameters. The column header contains the keyword when necessary, and the cell content is used as the parameter value. See Reading Spreadsheet using the Index Option for instance.
password = string (default = none)

Specifies a password for a protected spreadsheet file.

rWait = integer (default = 0)

Introduce a delay (in milliseconds) after opening a spreadsheet before accessing the data. This parameter can be used to work around an issue we encountered that Excel indicated it was not ready. The issue can arise during the data exchange with Excel Sheets that contain macros, plots or pivot tables where GDXXRW attempts to access a sheet while Excel is busy updating macros, graphs, and pivot tables.

checkDate (disabled by default)

When specified, no data will be written if the output file already exists and the file date for the output file is more recent than the file date for the input file. Provides a simple check to update the output file only if the input file has changed to save resources.

useRC (disabled by default)

Specify that all cell and range references use RC notation. So, instead of specifying the range Sheet1!A1:D6, one specifies Sheet1!R1C1:R6C4. When tracing is enabled, ranges will be reported in RC notation. This is a global option and applies to all cell references.

reCalc = flag (default = N)

Enable or disable the recalculations of cells inside Excel after writing to the spreadsheet. If there are many formulas in the spreadsheet the recalculation of cells can become very expensive and slowing down the writing process. By default, the recalculation is disabled and can be enabled via this option.

trace = integer (default = 1)

Sets the amount of information written to the log (for a better debugging). Higher values will generate more output. Valid range is 0 . . . 4.

0  Minimal information is included in the output
1  Message appears telling about each GDXXRW call indicating input file, output file and execution time
2  Message appears giving the level 1 output plus a listing for each symbol specified, indicating the type, sheet name, dimension, data range and the range of the row and column headers
3  Message appears giving the level 2 output plus cell ranges affected by reading, writing and clearing
4  Message appears giving the level 3 output plus cell addresses, and numerical or string values for every cell worked with

maxDupeErrors = integer (default = 0)

Sets the maximum number of duplicate records that is allowed when reading a spreadsheet and writing to a GDX file. The duplicate records for each symbol will be reported in the logfile, and if their accumulated number does not exceed the maximum specified using this option, the GDX file will not be deleted. This is a global option and applies to each symbol read from the spreadsheet.

The option is demonstrated in Reading Set from Lists with Duplication.
updLinks = integer (default = 0)

Specifies how links in a spreadsheet should be updated. The valid range is 0..3.

0  Doesn't update any references
1  Updates external references but not remote references
2  Updates remote references but not external references
3  Updates both remote and external references

runMacros = integer (default = 0)

This option controls the execution of the 'Auto_open' and the 'Auto_close' macros when opening or closing a spreadsheet. Valid range is 0..3.

0  Doesn't execute any macros
1  Executes Auto_open macro
2  Executes Auto_close macro
3  Executes Auto_open and Auto_close macro

6.29.2.2 Global Options

The following options affect the symbols that follow the option. They remain in effect unless they are redefined and used again for another symbol.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acronyms</td>
<td>0</td>
<td>Controls the handling of acronyms.</td>
</tr>
<tr>
<td>cMerge</td>
<td>0</td>
<td>Controls the handling of merged Excel ranges.</td>
</tr>
<tr>
<td>dSetText</td>
<td>N</td>
<td>Read explanatory text for set elements of domain sets.</td>
</tr>
<tr>
<td>epsOut</td>
<td>Eps</td>
<td>String to be used when writing the value for 'Epsilon'.</td>
</tr>
<tr>
<td>filter</td>
<td>0</td>
<td>Set the Excel filter for symbols written to Excel.</td>
</tr>
<tr>
<td>incRC</td>
<td>N</td>
<td>Include Excel row and column indices when a symbol is written to the GDX file.</td>
</tr>
<tr>
<td>mInfOut</td>
<td>-Inf</td>
<td>String to be used when writing the value for 'Negative infinity'.</td>
</tr>
<tr>
<td>NaIn</td>
<td>none</td>
<td>String to be used when reading a value for 'Not available'.</td>
</tr>
<tr>
<td>nameConv</td>
<td>N</td>
<td>Controls the interpretation of an Excel range.</td>
</tr>
<tr>
<td>NaOut</td>
<td>NA</td>
<td>String to be used when writing the value for 'Not available'.</td>
</tr>
<tr>
<td>pInfOut</td>
<td>+Inf</td>
<td>String to be used when writing the value for 'Positive infinity'.</td>
</tr>
<tr>
<td>resetOut</td>
<td>disabled</td>
<td>Reset the output strings for special values to their defaults.</td>
</tr>
<tr>
<td>squeeze</td>
<td>Y</td>
<td>Controls writing of default values of sub-fields of variables and equations resp. when reading from spreadsheet.</td>
</tr>
<tr>
<td>skipEmpty</td>
<td>1</td>
<td>Number of empty rows or columns indicating the end of a block when reading from spreadsheet using the top left cell specification.</td>
</tr>
<tr>
<td>UndfOut</td>
<td>Undf</td>
<td>String to be used when writing the value for 'Undefined'.</td>
</tr>
<tr>
<td>allUELs</td>
<td>Y</td>
<td>Controls the handling of UELs without associated values in the data range.</td>
</tr>
<tr>
<td>zeroOut</td>
<td>0</td>
<td>String to be used when writing the value for 'Zero'.</td>
</tr>
</tbody>
</table>
Some more detailed remarks on the global options:

**acronyms = integer (default = 0)**

A non-zero value indicates that acronyms can be expected and should be processed.

If no acronym processing takes place, reading an identifier in the data section of a sheet will generate an error. Writing an acronym to a sheet will write the internal numerical representation of the acronym.

Processing acronyms:

When reading a spreadsheet, an identifier in the data section of the sheet will be interpreted as an acronym and will be written to the GDX file.

When writing to a spreadsheet, a data tuple containing an acronym will be stored using the corresponding identifier of the acronym.

**cMerge = integer (default = 0)**

Option indicating how to read an empty cell that is part of a merged Excel range. See Reading merged Excel Ranges with cMerge. Possible values and their interpretation are:

0  Leave the cell empty
1  Use merged value in row and column headers only
2  Use merged value in all cells

**dSetText = flag (default = N)**

Valid only when reading a spreadsheet.

This controls the reading of explanatory text for set elements of domain sets. By default, no text is read for domain sets. If this option is activated this is changed. If an element appears more than once, the first one defines the explanatory text read.

**epsOut = string (default = Eps)**

String to be used when writing the value for 'Epsilon'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet and Writing Parameter to Spreadsheet including Zero Values.

**filter = integer (default = 0)**

Adds basic Excel filter to the columns of the spreadsheet to display only those values matching some conditions. Using this option when reading an Excel file will result in an error. Specifying filter=1 will set an Excel filter for the row of labels that are closest to the data values. When there are multiple rows in a column header (cDim > 1) we can specify filter=x where x is a number of the range 2..cDim, indicating to use a row farther away from the data values. See also Writing to Spreadsheet using a Filter.

**incRC = flag (default = N)**

Valid only when reading a spreadsheet.

Include Excel row and column indices when a symbol is written to the GDX file. For example, when we write a parameter P with indices I and J, without this option it will be written as P(I, J). When incRC is enabled, the parameter will be written as P(Excel_Rows, I, Excel_Columns, J). Note that the sets Excel_Rows and Excel_Columns will be added to the GDX file automatically.
mInfOut = string (default = -Inf)

String to be used when writing the value for 'Negative infinity'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

NaIn = string (default = none)

String to be used when reading a value for 'Not available'; this string is recognized in addition to the string 'NA' and is not case-sensitive. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

nameConv = flag (default = N)

Synonym: nC

The naming convention parameter is used to change the interpretation of an Excel range that does not contain an '!' (exclamation mark). For details see Excel Ranges below.

NaOut = string (default = NA)

String to be used when writing the value for 'Not available'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

pInfOut = string (default = +Inf)

String to be used when writing the value for 'Positive infinity'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

resetOut (disabled by default)

Reset the output strings for special values to their defaults. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

squeeze = flag (default = Y)

Synonym: sq

Writing to a spreadsheet:

The squeeze option affects the writing of sub-fields of variables and equations. A value for the field that is the default value for that type of variable or equation will not be written to the spreadsheet. For example, the default for .l (level value) is 0.0, and therefore zero will not be written to the spreadsheet. When we set squeeze=n, all values will be written to the spreadsheet.

The squeeze option for writing data is demonstrated in Reading Data from Spreadsheet and writing Data to Spreadsheet and Writing Parameter to Spreadsheet including Zero Values.

Reading a spreadsheet:

When the squeeze option is enabled, zero values for parameters will not be written to the GDX file. When the squeeze option is disabled, zero values will be written to the GDX file. In either case, empty cells, or cells containing blanks only, will never be written to the GDX file.

The squeeze option for reading data is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.
The `skipEmpty` option can be used when reading a spreadsheet and the range is specified using the top left corner instead of a block range. The value defines the number of empty row or column cells signal the end of a block. Valid values are 0..n. If the range is specified using a block range, `skipEmpty` will be ignored. Blank rows or columns will be skipped automatically.

Note that `skipEmpty` is also valid when using the `merge` resp. `clear` options in order to write data to spreadsheet (in a specific order determined by matching row and column labels already stored in the spreadsheet).

See Skipping Empty Rows and Columns or Reading Multi-dimensional Parameter from Spreadsheet for instance.

`UndfOut = string (default = Undf)`

String to be used when writing the value for 'Undefined'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

`allUELs = flag (default = Y)`

Valid only when reading a spreadsheet.

When enabled, all unique elements found in a range will be entered in the GDX file. When disabled, only those unique elements that are used in conjunction with a value will be entered in the GDX file.

`zeroOut = string (default = 0)`

String to be used when writing the value for 'Zero'; by default this is '0'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

### 6.29.2.3 Symbols

To write data to a spreadsheet or to a GDX file, one or more symbols and their associated Excel range need to be specified. See also Excel Ranges.

The general syntax for a symbol specification is:

```
dataType=symbolName {symbolAttributes}
```

Among the symbolAttributes, one specifies the `dataRange`, the `dimensions` of the symbol and some additional `symbolOptions` in general.

<table>
<thead>
<tr>
<th>dataTyp</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>Declare the symbol as parameter and define a individual name when reading from spreadsheet, or specify a parameter from a GDX file when writing to spreadsheet.</td>
</tr>
<tr>
<td>equ</td>
<td>Specify a sub-field of an equation from a GDX file when writing to spreadsheet.</td>
</tr>
<tr>
<td>var</td>
<td>Specify a sub-field of a variable from a GDX file when writing to spreadsheet.</td>
</tr>
<tr>
<td>set</td>
<td>Declare the symbol as set and define a individual name when reading from spreadsheet, or specify a set from a GDX file when writing to spreadsheet.</td>
</tr>
<tr>
<td>dSet</td>
<td>Declare the symbol as domain set and define a individual name when reading from spreadsheet, or specify a domain set from a GDX file when writing to spreadsheet.</td>
</tr>
</tbody>
</table>
**par = GAMS_Parameter**

Specify a GAMS parameter to be read from a GDX file and written to spreadsheet, or to be read from a spreadsheet and written to a GDX file.

When writing to a spreadsheet, special values such as Eps, Na and Inf will be written in ASCII. When reading data from a spreadsheet, the ASCII strings will be used to write the corresponding special values to the GDX file.

This datatype is used in the most examples. Reading parameters is demonstrated in Reading Parameter from Spreadsheet or Reading Multi-dimensional Parameter from Spreadsheet. Writing parameters from GDX to spreadsheet is demonstrated in Writing Parameter to Spreadsheet.

**equ = GAMS_Equation**

**var = GAMS_Variable**

A sub-field of a variable or equation can be written to a spreadsheet and should be specified as part of the symbolName. The fields recognized are .l (level), .m (marginal), .lo (lower bound), .up (upper bound), .prior (priority), and .scale (scale). The sub-field names are not case-sensitive. See also Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve.

A sub-field of a variable or equation cannot be read from a spreadsheet and written to a GDX file.

**set = GAMS_Set [values = valueType]**

In GAMS we can define a set by specifying all its elements. In addition, each tuple can have an associated text. To read a set from a spreadsheet, the values option is used to indicate if there is any data, and if there is, if the data should be interpreted as associated text or as an indicator whether the tuple should be included in the set or not.

Reading sets is demonstrated in Reading Set from Spreadsheet or Reading Set Elements associated with Data or Text using the values Option (focusing on the values option) for instance. Writing sets from GDX to spreadsheet is demonstrated in Writing Set to Spreadsheet.

<table>
<thead>
<tr>
<th>valueType</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Based on the range, row and column dimensions for the set, the program decides on the valueType to be used. This is the default for values.</td>
</tr>
<tr>
<td>noData</td>
<td>There is no data range for the set; all tuples will be included.</td>
</tr>
<tr>
<td>YN</td>
<td>Only those tuples will be included that have a data cell that is not empty and does not contain '0', 'N' or 'No'.</td>
</tr>
<tr>
<td>sparse</td>
<td>Only those tuples will be included that have a data cell that is not empty. The string in the data cell will be used as the associated text for the tuple.</td>
</tr>
<tr>
<td>dense</td>
<td>All tuples will be included. A string in the data cell will be used as the associated text for the tuple.</td>
</tr>
</tbody>
</table>

Due to backward compatibility, valueType=string or all are also recognized and are synonyms for valueType=dense. The following table summarizes which valueType will be used when reading a set, if a valueType was not specified:
When writing to a spreadsheet, the entire set is written to the spreadsheet and the writing of the associated text is governed by the values option:

<table>
<thead>
<tr>
<th>valueType</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>If rDim=0 or cDim=0, auto means string, otherwise auto means YN.</td>
</tr>
<tr>
<td>noData</td>
<td>Neither associated text nor 'Y' is written for a set element.</td>
</tr>
<tr>
<td>YN</td>
<td>A 'Y' is written for a set element.</td>
</tr>
<tr>
<td>string</td>
<td>The associated text is written for a set element. If no text is</td>
</tr>
<tr>
<td></td>
<td>stored with set element the cell will be empty.</td>
</tr>
</tbody>
</table>

Due to backward compatibility, valueType=dense, sparse or all are also recognized and are synonyms for valueType=string.

dSet = GAMS_Set

A domain set is used to read the domain of a set from a spreadsheet row or column. Either the row or the column dimension (rDim or cDim) should be set to '1' to specify a row or column for the set, resulting in a one-dimensional set. Duplicate labels in the range specified do not generate an error message. For instance, see also Reading Set from Lists with Duplication.

Note that reading explanatory text of set elements is not supported by dSet. In order to read explanatory text, use set instead. If there are duplicate set element labels in your data, use the set symbol specification while increasing the value of the immediate option maxDupeErrors to oppress an error message when reading duplicates.

text = "String of characters" {dataRange}
textID = Identifier {dataRange}

Write the text to the cell specified in the DataRange. In addition, textID will write the explanatory text of the Identifier in the cell to the right of the dataRange.

A Text directive can be followed by a link=Address or linkID=identifier directive. Using link will create a hyperlink to an external page or to a cell in the spreadsheet, while linkID will create a hyperlink to the top left corner of the symbol specified. See Writing to Spreadsheet adding Text and Hyperlinks for instance.

hText = "String of characters" {dataRange}
vText = "String of characters" {dataRange}

Write a string of characters in the horizontal direction for hText or vertical direction for vText. Text for the next cell is indicated by a comma. In order to write a comma as part of the text, the comma needs to be preceded by a backslash. See Writing Set to Spreadsheet for instance.

6.29.2.3.1 Symbol Attributes  The following options apply to the symbol preceding the option, and only affect that symbol:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataRange</td>
<td>Cell A1 of the first sheet</td>
<td>Specify the Excel range of the symbol for reading from spreadsheet or for writing to spreadsheet.</td>
</tr>
<tr>
<td>dim</td>
<td>2 when reading from spreadsheet</td>
<td>Total dimension of the symbol.</td>
</tr>
<tr>
<td></td>
<td>Defined by the symbol dimension stored in the GDX file when writing to spreadsheet</td>
<td></td>
</tr>
<tr>
<td>cDim</td>
<td>1</td>
<td>Column dimension of the symbol.</td>
</tr>
<tr>
<td>rDim</td>
<td>dim-1</td>
<td>Row dimension of the symbol.</td>
</tr>
<tr>
<td>merge</td>
<td>disabled</td>
<td>When enabled, the data will be written in a specific order determined by matching row and column labels already stored in the spreadsheet.</td>
</tr>
<tr>
<td>clear</td>
<td>disabled</td>
<td>In addition to the effect of merge, already existing values in the data range of the spreadsheet are removed before writing.</td>
</tr>
<tr>
<td>colMerge</td>
<td>0</td>
<td>Determines the columns for which non-empty content of the previous cell will be used as content for the empty cell of a column.</td>
</tr>
<tr>
<td>intAsText</td>
<td>Y</td>
<td>Determines the cell format when writing unique elements that are a proper integers to spreadsheet.</td>
</tr>
<tr>
<td>ignoreRows/Cols</td>
<td>none</td>
<td>Specify rows and columns to be ignored when reading from spreadsheet.</td>
</tr>
</tbody>
</table>

Some more detailed remarks on the symbol attributes:

**dataRange**

\[ \text{rng} = \text{Excel Range} \]

The Excel Range for the data for the symbol. Note that an empty range is equivalent to the first cell of the first sheet.

**dimensions**

\[ \text{dim} = \text{integer} \]

The total dimension for the symbol.

\[ \text{cDim} = \text{integer} \]

Column dimension: the number of rows in the data range that will be used to define the labels for the columns. The first \( c\text{Dim} \) rows of the data range will be used for labels.

\[ \text{rDim} = \text{integer} \]
Row dimension: the number of columns in the data range that will be used to define the labels for the rows. The first $r\text{Dim}$ columns of the data range will be used for the labels.

More about dimensions:

When reading data from a GDX file and writing to a spreadsheet, the dimension of the symbol is known. When reading a spreadsheet and writing to a GDX file, the dimension is not known.

The sum of $c\text{Dim}$ and $r\text{Dim}$ determine the dimension of the symbol. This dimension is used when writing data to a GDX file, and is used to verify the dimension of a symbol when reading from a GDX file.

When reading a GDX file, the dimension of a symbol is known, and therefore the $c\text{Dim}$ or $r\text{Dim}$ parameter can be omitted. If both $c\text{Dim}$ and $r\text{Dim}$ are omitted, the program assumes that $c\text{Dim}=1$ and $r\text{Dim}=\text{dim}-1$.

symbolOptions

The options below are only valid when reading a GDX file and writing to a spreadsheet.

By default, writing data to a spreadsheet will include the row and column labels in addition to the data. The row and column labels will appear in the same order as they appear in the GDX file.

merge (disabled by default)

Using the merge option assumes that the row and column labels are in the spreadsheet already. For each value read from the GDX file, the location of the row and column labels is used to update the spreadsheet. Using the merge option will force the data to be presented in a given order using the row and column labels. Spreadsheet cells for which there is no matching row/column pair will not be changed. The matching of labels is not case-sensitive. See also Writing to Spreadsheet with merge Option Example.

Note that the skipEmpty option value affects the reading of the row and column labels from spreadsheet in case of top left range specification (while skipEmpty is ignored in case of block range specification).

Warning: The merge or clear option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.

clear (disabled by default)

The clear option is similar as the merge option, except that the data range will be cleared before any data is written. See also Writing to Spreadsheet with clear Option Example.

colMerge = integer (default = 0)

The number of columns that will use a previous value in that column if the cell is empty. Can only be used when reading from a spreadsheet. See Reading empty Cells with colMerge.

intAsText = flag (default = Y)
Unique elements that are a proper integer can be written as text or as an integer value. The default is Y, which will write the unique element as a string. Note that this impacts the sorting order and can be used when using an Excel filter on a data range.

**Ignoring Rows and Columns when reading from a spreadsheet**

```plaintext
ignoreRows = rownr, rownr, rownr:rownr
ignoreColumns = colnr, colnr, colnr:colnr
```

Row numbers are represented by integers. Column numbers are represented by Excel column numbers, like A, CD, IV etc, or by integers.

**Note**

- Ignoring rows or columns is only allowed when reading a spreadsheet.
- The specification of ignored rows or columns follows the symbol specification and only applies to that symbol.
- When ignoring a column that would be part of an index if the column was not ignored, the range for the index will be extended for each column that is ignored. The same holds for ignored rows that are part of an index.

See also **Ignoring Rows and Columns**.

### 6.29.2.4 Syntax Elements

The most options are specified by using an integer, a string or a flag. Note that the options `useRC`, `resetOut`, `checkDate`, `merge` and `clear` are enabled or disabled by simply adding the keyword to your GDXXRW statement.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>An unsigned integer</td>
</tr>
<tr>
<td>string</td>
<td>A string of characters; a string can be quoted with single or double quotation marks.</td>
</tr>
</tbody>
</table>
| flag | True values: 1, Y or Yes  
False values: 0, N or No  
(not case-sensitive) |

### 6.29.2.5 Excel Ranges

An Excel Range is specified using the standard Excel notation: `SheetName!CellRange`.

When the `SheetName!` is omitted, the first sheet will be used. A CellRange is specified by using the `TopLeft:BottomRight` cell notation like `A1:C12`. When `BottomRight` is omitted, the program will extend the range as far down and to the right as possible. (Using '..' instead of ':' is supported.)

Excel also allows for named ranges; a named range includes a sheet name and a cell range. Before interpreting a range parameter, the string will be used to search for a pre-defined Excel range with that name. See **Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges** for instance.
When writing to a spreadsheet and a sheet name has been specified that does not exist, a new sheet will be added to the workbook with that name. Reading a spreadsheet and using an unknown range or sheet name will result in an error.

The following table summarizes all possible input combinations and their interpretation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Sheet used</th>
<th>Cell(s) used</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>First sheet</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>First sheet</td>
<td>Name</td>
<td>When nc=0</td>
</tr>
<tr>
<td>Name!</td>
<td>Name</td>
<td>A1</td>
<td>When nc=1</td>
</tr>
<tr>
<td>!Name</td>
<td>First sheet</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Name1!Name2</td>
<td>Name1</td>
<td>Name2</td>
<td></td>
</tr>
</tbody>
</table>

The term nc= refers to the nameConv option.

### 6.29.3 Return Codes

On success, GDXXRW will return 0 as error code. However, there might be an error which will be signaled with a specific return code in addition to an error message.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>Cannot write log</td>
</tr>
<tr>
<td>2</td>
<td>GDX error</td>
</tr>
<tr>
<td>3</td>
<td>No input file</td>
</tr>
<tr>
<td>4</td>
<td>Input file not found</td>
</tr>
<tr>
<td>5</td>
<td>Bad parameter</td>
</tr>
<tr>
<td>6</td>
<td>Read error</td>
</tr>
<tr>
<td>7</td>
<td>Problem loading GDX DLL</td>
</tr>
<tr>
<td>8</td>
<td>Symbol not found</td>
</tr>
<tr>
<td>9</td>
<td>Dimension different</td>
</tr>
<tr>
<td>10</td>
<td>Types different</td>
</tr>
<tr>
<td>11</td>
<td>Bad UELs</td>
</tr>
<tr>
<td>12</td>
<td>Bad output file</td>
</tr>
<tr>
<td>13</td>
<td>Problem opening Excel</td>
</tr>
<tr>
<td>14</td>
<td>Problem writing to Excel</td>
</tr>
<tr>
<td>15</td>
<td>Problem reading from Excel</td>
</tr>
<tr>
<td>16</td>
<td>Duplicate entry</td>
</tr>
<tr>
<td>17</td>
<td>Cannot add sheet</td>
</tr>
<tr>
<td>18</td>
<td>Bad cell value</td>
</tr>
<tr>
<td>19</td>
<td>Dimension conflict</td>
</tr>
<tr>
<td>20</td>
<td>Data exceeds range</td>
</tr>
<tr>
<td>21</td>
<td>Exceeds range or memory problem</td>
</tr>
<tr>
<td>22</td>
<td>Deprecated</td>
</tr>
</tbody>
</table>
### Return Code Interpretation

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Program aborted</td>
</tr>
<tr>
<td>24</td>
<td>Merge range empty</td>
</tr>
<tr>
<td>25</td>
<td>Too many columns skipped</td>
</tr>
<tr>
<td>26</td>
<td>Too many rows skipped</td>
</tr>
</tbody>
</table>

### 6.29.4 Warning

When executing `GDXXRW` twice and redirecting output to the same log file may result in a fatal error. For example:

```
gdxxrw step1 parameters > logfile
gdxxrw step2 parameters > logfile
```

The execution of step2 may fail, because Excel will close the logfile in step1 in a delayed fashion, but return control to `GDXXRW` immediately. Using the `log` or `logAppend` parameter will avoid this problem.

### 6.29.5 Reading from Spreadsheet - Examples:

#### 6.29.5.1 Reading Set from Spreadsheet

Assuming we want to read set elements from the first sheet of the spreadsheet file `exampleData.xlsx` and write the data to `exampleData.gdx`.

Either of the following two statements below reads the second row of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx set=i1 rng=readingSets!A2:C2 cDim=1
gdxxrw exampleData.xlsx dSet=i1a rng=readingSets!A2:C2 cDim=1
```

When the output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. Since all elements in the second row are unique, there is no need of increasing the `maxDupeErrors` parameter to avoid an error message when defining the symbol as `set`. By specifying the symbol directly as a `dSet` (domain set) in the second statement, duplicate labels would be removed without throwing an error. We set `cDim` to one so that the first row of the range is used for the labels of the set.

On the other hand if we want to read set elements listed in a column:

Either of the following two statements reads column A of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx set=j1 rng=readingSets!A35:A37 rDim=1
gdxxrw exampleData.xlsx dSet=j1a rng=readingSets!A35:A37 rDim=1
```

Besides the range we also had to change the parameter `rDim` to indicate that the first column of the range is to be used for the labels.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample18] for reference.
6.29.5.2 Reading Set and Explanatory Text

Suppose we want to read the set elements in the ninth row and their associated text in the tenth row of the following spreadsheet:

We can read the set elements and their explanatory text by executing the command:

```plaintext
gdxxrw exampleData.xlsx set=i3 rng=readingSets!A9:E10 cDim=1
```

To read the explanatory text, we simply include the tenth row within the range of the symbol `i3` and specify `cDim=1`. By doing this, the first row of the range, i.e. the elements of the ninth row, will be used as the set elements, while the tenth row will be interpreted as their associated text (depending on the `values` option specified. By default, the `values` option is set to `dense` in this example, i.e. all elements will be included and the string in the associated data cell will be used as explanatory text. See also Reading set elements associated with Data or Text for instance.).

Note here the explanatory text of the set element `skipme2` is just a `Y` as it has a blank entry for the explanatory text.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample18] for reference.

6.29.5.3 Reading Set Elements associated with Data or Text using the values Option

When reading set elements from spreadsheet, the `values` option can be used to control whether elements and associated text are included in the set or not. We use the data displayed in the spreadsheet below to demonstrate the `yn`, `dense`, `sparse` and `noData` specifications:

The set element names are stored in the first row, the associated data cells in the second row.

1. values=yn
   Run the following command to load those element names associated with nonzero data or yes without storing the data as explanatory text:

   ```plaintext
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=yn
```

   The set `A` will contain the elements `a, b, e, f, h, i, k` and `m`, since the elements `c, d, g, j` and `l` are associated with a zero, a blank or a no resp. N (case insensitive).

2. values=dense
   This option must be specified, if we want to read all elements while using the strings in the data cells as explanatory text.

   ```plaintext
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=dense
```

3. values=sparse
   To read in all elements having a non-empty data cell while interpreting the string in the data cell as explanatory text, run the following command:
The set $A$ will contain all elements except for $j$, since the associated data cell is empty.

4. **values=**noData

This option must be used, if we want to read all set elements while ignoring the data range. Especially, the data cells will not be interpreted as explanatory text. To read the elements from the range A1:M1, one could run the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1 values=noData
```

While all elements $a$ - $m$ will be included this way, the second row will be automatically interpreted as explanatory text (even though we specified only the first row within the `rng` statement). This might be not desirable at all in some situations, e.g. when reading the city names in the next example **Reading Set from Data Tables**, we do not want to have the numbers 5000, 6000 and 0 to be explanatory text for the city names. Run the following command to include all elements in your set without interpreting the cells in the second row as explanatory text:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1 values=noData
```

5. **values=**auto (default)

The second table within the description of the `values` options indicates which value type will be used by default based on the `rng`, `cDim` and `rDim` specifications for the set. For instance, when running the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1
```

the value type used is **dense**, since we specified a block range with a data row and `rDim` equals zero.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample9] for reference.

### 6.29.5.4 Reading Set from Data Tables

One may wish to load set elements from a data table. Given a spreadsheet segment like the following:

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cleveland</td>
<td>chicago</td>
<td>dallas</td>
</tr>
</tbody>
</table>
```

We can take the set across the top containing the elements cleveland, chicago and dallas with any of the following commands:

```
gdxxrw exampleData.xlsx set=i6 rng=readingSets!B20:D20 cDim=1 values=noData
gdxxrw exampleData.xlsx dSet=i6a rng=readingSets!B20:D20 cDim=1
gdxxrw exampleData.xlsx set=i6c rng=readingSets!B20:D21 cDim=1 values=noData
```

Note the usage of the `values` option in order to avoid reading the numbers as explanatory text. See also Reading set elements associated with Data or Text. However, this can also be avoided by declaring the symbol as a domain set using the symbol declaration `dSet`.

We can also take a set vertically from column A as follows:

```
gdxxrw exampleData.xlsx dSet=j4 rng=readingSets!A21:A23 rDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample18] for reference.
6.29.5.5 Reading Set from Lists with Duplication

One may wish to extract set elements from a spreadsheet where there is no unique list of elements that can be read but rather a list where some names are repeated. In the example below note that in rows 26 and 27 there are set element names but they are duplicated:

One can read this using dSet as follows:

```
gdxxrw exampleData.xlsx dSet=i7 rng=readingSets!B26:E26 cDim=1 dSet=i8 rng=readingSets!B27:E27 cDim=1
```

Both domain sets will be read within a single GDXXRW call. The `rng` and `cDim` specifications affect only the symbol that they are following directly.

It may be favored in some situation to use the `set` symbol instead (e.g. for reading explanatory text). To oppress an error message when reading sets with duplication, one must specify a sufficient large number within the `maxDupeErrors` option.

```
gdxxrw exampleData.xlsx maxDupeErrors=4 set=i7 rng=readingSets!B26:E26 cDim=1 values=noData set=i8 rng=readingSets!B26:E26 cDim=1 values=noData
```

For the data in this example, four is a sufficient large number since there are two duplicates for the first and two duplicates for the second set within each range. Note the usage of the `values` option in order to avoid reading 'cleveland' as explanatory text for the elements of set `i7` and to avoid reading the numbers as explanatory text for the elements of `i8`. See also Reading set elements associated with Data or Text for more informations about the `values` option.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample18] for reference.

6.29.5.6 Reading Parameter from Spreadsheet

Assuming we want to read parameter `data1` from the file Test1.xlsx and write the data to Test1.gdx.

The following statement reads parameter `data1` from the spreadsheet above (using the `par` data type):

```
gdxxrw Test1.xlsx par=data1 rng=A1:D3 cDim=1 rDim=1
```

The sheet name in a range can be omitted when it refers to the first sheet. The elements in the first row and first column of the data range will be used as labels for the two dimensional parameter `data1` by defining `cDim=1` and `rDim=1` (see also `cDim` resp. `rDim`).

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample5] for reference.
6.29.5.7  Reading Parameter from Spreadsheet with Duplication

The same data as in the previous example, but organized differently. We use the dSet symbol specification instead of set to read set I (in column A) and set A (in column B), since there are duplicate entries in column A resp. column B.

The following statement reads parameter data2 from the spreadsheet above:

```
gdxxrw Test1.xlsx par=data2 rng=EX2!A1 rDim=2 dSet=I rng=EX2!A1 rDim=1 dSet=A rng=EX2!B1 rDim=1
```

By setting rDim=2 for the parameter data2 we indicate to use the first two columns of the data range as the labels for the parameter values. Since the sheet does not contain further data, one can specify the ranges using the top left cell notation without hesitation.

When using a few symbols, the command line can become too long to be practical. In such case, use a text file to hold the parameters. A parameter file can contain multiple lines to increase readability and a line starting with a '∗' will be ignored.

```
* file example6.txt
par =data2 rng=EX2!A1 rDim=2
dSet=I   rng=EX2!A1 rDim=1
dSet=A   rng=EX2!B1 rDim=1
```

An option file is indicated by preceding the file name with a @ (At sign.).

```
gdxxrw Test1.xlsx @example6.txt
```

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample6] for reference.

Note

- An option file can contain multiple lines to increase readability.
- When reading parameters from a text file, lines starting with an asterisk (∗) will be ignored and act as a comment.
- An option file can also be written during the execution of a GAMS model using the GAMS Put Facility and the subsequent GDXXRW commands must use execute command so the put file is written before it is to be read (this would happen when using the compile time command $call to run GDXXRW).
- Using an option file can be useful in reducing execution time by combining the options and symbols from multiple GDXXRW calls in a single option file used in a single GDXXRW call.
6.29.5.8 Reading Multi-dimensional Parameter from Spreadsheet

This example illustrates how to read a four dimensional parameter from spreadsheet:

The strings in the first two columns and the first two rows of the data range A1:F6 shall be used as labels for the parameter values. Therefore, we define \( r\text{Dim}=2 \) and \( c\text{Dim}=2 \) (see also \( r\text{Dim} \) and \( c\text{Dim} \)). The parameter will be named \text{data3} \ (within the \text{par} declaration). Run the following command to read the data with \text{GDXXRW}:

\[
gdxxrw Test1.xlsx \text{ par=data3 rng=EX3!A1:F6 rDim=2 cDim=2}
\]

Note that the data range was specified using the block range notation. However, it might be more comfortable to specify only the top left corner sometimes, but empty rows or columns may affect the reading process, i.e. \text{GDXXRW} might stop to early when encountering empty rows or columns or it will try to read data separated by empty rows or columns not being part of the data you wish to read. When we specify the range as a block, an empty row or column will always be ignored. When we specify the top left cell only, the \text{skipEmpty} option can be used to ignore one or more empty rows or columns. However, for the data in this example, we do not need to ignore empty rows or columns within the data range, but suppose there is non-relevant data starting in column H. By default (\text{skipEmpty}=1), \text{GDXXRW} would try to read the data starting in H. When we specify \text{skipEmpty}=0 and cells A7, B7, G1 and G2 are empty, the range can be specified with the top left cell only in this example:

\[
gdxxrw Test1.xlsx \text{ skipEmpty=0 par=data3 rng=EX3!A1 rDim=2 cDim=2}
\]

Since \text{skipEmpty} is a global option, affecting every symbol that follows, we define it before declaring the parameter \text{data3}.

This example is also part of the GAMS Data Utilities Library, see model \[\text{GDXXRWExample7}\] for reference.

6.29.5.9 Reading Spreadsheet using the Index Option

The \text{index} option is used to read a number of parameters and sets based on information stored in the spreadsheet itself. By doing this, the \text{GDXXRW} command becomes quit short and the informations on how to read the data can be written directly within the spreadsheet to increase readability. The first row of the range is used for column headings indicating additional parameters. We will discuss the results only briefly, because all other options used in this example were demonstrated before.

Suppose we want to read the parameters and sets from the following spreadsheet:

The informations about all parameters and sets are stored within the sheet index in the same spreadsheet file exampleData.xlsx as testIndex above:

The following statement reads parameters and sets from the spreadsheets using the \text{index} option:

\[
gdxxrw exampleData.xlsx output=gdxAll.gdx index=index!A1
\]

We use the \text{output} option to write the data to gdxAll.gdx for demonstration here. By default, \text{GDXXRW} would write to testIndex.gdx. Some brief remarks on the results:
• Elements of set i1 and i1a: trains, cars and planes. There is no explanatory text, since the data range (third row) is empty. We can use dim=1 equivalently to cDim=1 in this example, because by default, cDim=1 and rDim=dim-1. See also dimensions.

• When reading the sets i2 and i3, we are defining which elements and explanatory text will be stored by specifying the values option.

• The set i4 will contain all elements of the thirteenth row. Afterwards, skipEmpty is set to zero, affecting the reading of i4a and i5. Unlike i4, the sets i4a and i5 will not contain the element houston, since skipEmpty=0 and the empty column signals GDXXRW to stop reading (note that the range is not defined using the block range specification). Since there is a data range for i5, the strings city1 - city5 will be used as explanatory text by default. After reading i4a and i5, skipEmpty is reset to zero (default).

• Each of the sets i6, i6a and i6b will contain the elements cleveland, chicago and dallas. There is no explanatory text for any of the sets in the GDX file, however, one must enforce this for the set i6 with values=noData.

There are no interesting details for the remaining sets and parameters to discuss here. The complete example and the results displayed within GAMS can be found in the GAMS Data Utilities Library, see model [GDXXRWExample10] for reference.

Note

• The parameters and sets are read using the specifications within the myIndex sheet. They are treated as if they appeared directly on the command line.

• In the spreadsheet, the first three columns of the range have a fixed interpretation: DataType (par, set, dSet, equ, or var), Item name identifier and spreadsheet data range. The fourth and following columns can be used for additional parameters like dim, rDim, cDim, merge, clear and skipEmpty. The column header contains the keyword when necessary, and the cell content is used as the option value.

• When an entry appears in a column without a heading then it is directly copied into the GDXXRW option file. Thus in the example above the items in column G are directly copied into the file.

• Rows do not need to have entries in the first three columns if one just wants to enter persistent options such as skipEmpty or some of the special character string re-definitions (as in row seven and ten from the spreadsheet above).

### 6.29.5.10 Reading Data from Spreadsheet and Loading into GAMS

One can use $call to execute the GDXXRW command in the GAMS code to read from spreadsheet at compilation time (the data is taken from the previous example):

```gams
$call gdxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
```

Getting a set from the spreadsheet into a GDX file is only half the battle. One must also use commands in GAMS to load the data as discussed in the chapter Using GAMS Data Exchange or GDX Files. At compile time this is done using:
Set i9;
* read/load set from data at compile time
$call gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
$gdxIn testIndex.gdx
$load i9

where the set must be declared in a set statement then one can if needed create the GDX file using GDXXRW, then one uses a $gdxIn to identify the source file and a $load to bring in the data.

Some users may wish to load sets at execution time. However, this is limited to subsets that are dynamic sets and cannot be used in domains. To do this one simply uses the statements as above, but substitutes execute in place of $call as follows:

Set i9(i6a);
* read/load set from data at execution time
execute 'gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData'
execute_load 'testIndex' i9;

where the set i9 must be declared as a subset in a set statement (of i6a in this case), then one can if needed create the GDX file using execution time GDXXRW, and an execute_load to bring in the data with an identification of the GDX source file name. Note that we used the set i6a as superset, fitting best to the data from the previous example. However, one could also use the statement Set i9(*);

One can load the universe of labels from a GDX file into a set at run-time using the syntax:

execute_load 'someFile', someSet=*

Note
In doing this, only labels known to the GAMS program will be loaded.

6.29.5.11 Reading empty Cells with colMerge

Suppose we want to read the four dimensional parameter from the following spreadsheet:

The cells B4, B5, C4 and D5 might be empty to avoid duplication, i.e., the non-empty content of the previous cell in the same column shall be used as content for the empty cell. In particular: the content of B3 shall be used for the content of B4 and B5, the content of C3 for C4 and the content of D4 for cell D5. Reading the above spreadsheet using the following GAMS statement:

```
gdxxrw exampleData.xlsx par=A_d rng=colMerge!B2 rDim=3 cDim=1
```

results in empty cells B4, B5, C4 and D5, causing troubles if you want to declare the parameter as A(number,number,number,color) certainly:

Adding the symbol attribute colMerge, we use the non-empty content of the previous cell in the same column as the content for the empty cell. Specifying colMerge=2 will do this for the first two columns for instance.

```
gdxxrw exampleData.xlsx par=A_2 rng=colMerge!B2 rDim=3 cDim=1 colMerge=2
```

Only the two entries corresponding to the cell D5 are still empty, since we do not specify all three columns within colMerge:
Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an `<empty>` entry instead.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample20] for reference.

6.29.5.12 Reading merged Excel Ranges with cMerge

Suppose we want to read the three dimensional parameter from the following spreadsheet:

Note that the label 'red' is centered over the merged cells B1, C1 and D1 and label 'green' over the merged cells E1 and F1. Additionally, in the data range, the cells B3:C3 and B4:C4 are merged, too. The option cMerge can be used to control the way merged cells are handled. We will discuss the effect of the possible values for cMerge on the data presented above by running the following commands (an option file is used to increase readability):

```
$onEcho > howToRead.txt
cMerge=0 par=B_d rng=cMerge!A1 rDim=1 cDim=2
cMerge=1 par=B_1 rng=cMerge!A1 rDim=1 cDim=2
cMerge=2 par=B_2 rng=cMerge!A1 rDim=1 cDim=2
$offEcho
$call gdxxrw exampleData.xlsx output=cMerge.gdx @howToRead.txt
```

We specify cMerge in advance, since it is a global option affecting every symbol that follows. Executing the three statements will create three different output files, all displayed in GAMS Studio:

Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an `<empty>` entry instead.

Some remarks on the results:

**cMerge=0** (default)
Empty cells being part of a merged Excel range will remain empty. Thus, the cells C1, D1, F1, C3 and C4 from the spreadsheet above will remain empty when reading with GDXXRW. Since C3 and C4 are empty while being part of the data range, they won’t show up in the GDX file. The values in the last column of the GDX file are useful to compare the results with the spreadsheet. For instance, the UELs of the value 5 are a, `<empty>` and two, since the cell F1 is empty, while A3 contains the string a and F2 the string two.

**cMerge=1**
The value of a merged range within a row or column header will be used for all cells being part of the merged range. Thus, the string 'red' will be used for the cells in the column header C1 and D1 and the string 'green' for the cell F1. Since C3 and C4 are part of the data range, they will remain empty and are not displayed in the GDX file. As you can see, there is no longer an empty UEL.
cMerge=2
The value of a merged range will be used for all cells being part of the merged range, i.e. cMerge=1 is extended to the data range. Therefore, the value 1 resp. 11 will be also used for the cell C3 resp. C4, appearing in the GDX file for the first time. Since there is no change in handling merged cells within the row or column header, all values have non-empty labels.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample20] for reference.

6.29.5.13 Skipping Empty Rows and Columns

By using the skipEmpty option, we can control the way blank rows or columns are handled and causes GDXXRW to either stop or skip over if a blank row or column is encountered when using the top left corner range specification instead of a block range. If the range is specified using the TopLeft:BottomRight cell notation (often refered as block range notation), empty rows or columns will be skipped automatically. Suppose the data is stored in the following spreadsheet:

We can read this spreadsheet and skip blank rows and columns with the following command:

gdxxrw exampleData.xlsx par=A_d rng=skippingRC!A2 rDim=2 cDim=1

or

gdxxrw exampleData.xlsx se=1 par=A_1 rng=skippingRC!A2 rDim=2 cDim=1

Note that there will be no difference concerning the generated GDX files, since skipEmpty is set to one by default. On the other hand, if skipEmpty is set to zero

gdxxrw exampleData.xlsx se=0 par=A_0 rng=skippingRC!A2 rDim=2 cDim=1

the blanks terminate the read not reading the rail column and the san francisco.chicago row. After loading into GAMS the data become:

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample21] for reference.

Note
The skipEmpty option must appear before any par, set, dSet etc statements that use it and will persist for the rest of the statements in a command unless it is set to another value.
6.29.5.14 Ignoring Rows and Columns

We can use the options ignoreColumns and ignoreRows to ignore columns and rows when reading data for a symbol. Suppose we want to ignore the red colored rows and columns of the following spreadsheet, when reading the four dimensional parameter:

We can read this spreadsheet and ignore the red colored columns and rows with GDXXRW by running the following command:

```
gdxxrw exampleData.xlsx par=A rng=ignoringRCC1 cDim=2 rDim=2 ignoreRows=2,6 ignoreColumns=D,G
```

The options ignoreRows and ignoreColumns are symbol options and therefore must appear after the symbol specification, affecting only this particular symbol.

In the example above we ignored column D which would have been part of the index for the rows. So the range for the row index was extended with column E. The E column is no longer part of the data range. The treatment of the column index is similar. The second row would have been part of the column index, and now that the row is ignored, the next row becomes part of the column index and the third row is no longer part of the data range.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample21] for reference.

Note

The ignoreColumns and ignoreRows options appear after any par, set, dSet etc GDXXRW command instruction and only affect reading of that item.

6.29.5.15 Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges

As mentioned in section Excel Ranges, the range of a symbol to be read can be defined by using named ranges. To name a cell range within Excel, simply select the cell range and type in the name you want to assign to this particular range by using the name-box next to the top left corner of your spreadsheet. Suppose we want to read the data in the range A1:D3 taken from the example Reading Parameter from Spreadsheet:

Instead of specifying the range explicitly by rng=A1:D3, we use the pre-defined named range 'parRange', i.e. rng=parRange. GDXXRW uses the string specified to search for a matching pre-defined named range first. In summary, run the following command to read the parameter data4 from the file Test1.xlsx:

```
gdxxrw Test1.xlsx par=data4 rng=parRange rDim=1 cDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample5] for reference.

Note

In Excel, one can assign a single name to several separated block ranges, e.g. assign the name 'disconnected' to the block ranges A1:D3 and F4:H5 (by holding 'Strg' while selecting the second block range and using the name-box to assign the name). However, such disconnected data cannot be read using the named-range specification.
6.29 GDXRW

6.29.6 Writing to Spreadsheet - Examples:

6.29.6.1 Unloading Data from GAMS before Writing to Spreadsheet

When writing to spreadsheet with GDXRW, one must use commands in GAMS to place the data into the GDX file at first (see also Using GAMS Data Exchange or GDX files). When reading data, it is often desirable to use a $call command to run GDXRW and the statements $gdxIn and $load afterwards to load the data, allowing also domain definitions (at compile time) for instance. This is hardly ever desirable or realizable when unloading and writing to a spreadsheet, for example, if one wish to write the solution after solving a model to spreadsheet. This should generally not be done at compile time so one should only use the execute and execute_unload commands at execution time as follows:

execute_unload 'test.gdx' someParameter;
execute 'gdxxrw test.gdx par=someParameter rng=A1'

where the execute_unload tells what data to place in the GDX file and determines the GDX source file name. The matching GDXRW execution tells the name of the GDX file, the name of the spreadsheet (optional) and identifies the data to write.

See also Writing Set to Spreadsheet and Writing Parameter to Spreadsheet for demonstration.

Note

- One must be careful when using GDXRW as each time the command is executed the GDX file is erased and only has the current contents and thus should be written just before if reusing the name.
- One also obtains output of sets using the command execute_unload where the GDX file automatically includes all sets associated with unloaded parameters, variables and equations without need to list the set element names.

6.29.6.2 Writing Set to Spreadsheet

At first, we will create a GDX file containing a simple set using the execute and execute_unloading directives. Most of the elements have an explanatory text:

Set x / element1 'explanatory text'
                 element2
                 element3 'previous element does not have explanatory text' /;

execute_unload 'writingSet.gdx' x;

Of course, in this particular code section above, one could also use $gdxOut and $unload. The values option can be used to control whether explanatory text is written to the spreadsheet. We'll demonstrate all three possible values explicitly within a single GDXRW execution:
Before executing this example, check if the Excel file (writingSet.xlsx) is open. If you run GDXXRW for writing a file sharing conflict will arise. To avoid this problem, either close the Excel file or use the Excel Tools menu to make this a shared notebook. After writing to the spreadsheet (still opened), use the Excel "File Save" command to verify the changes made.

By adding two additional commas within the hText statement, the cells B1, C1 and E1, F1 will be skipped when writing the text to the first row of the spreadsheet. If values=noData, neither explanatory text nor a Y are written to spreadsheet for the set elements. If values=yn, GDXXRW writes a Y for each set element to spreadsheet. To write the explanatory text, specify values=string.

Since cDim=0, the default option is string (see values). Imagine a two dimensional set, one could write the set in a table format, i.e. cDim=1 and rDim=1. By default, GDXXRW would write this set using the values=yn format.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample11b] for reference.

Note
- A workbook cannot in general be open unless you have made special provisions with an error signaled indicating a file sharing conflict will arise when the target file is open in Excel.
- To avoid the sharing conflict error the user must either close the file or indicate that the spreadsheet is a shared Excel workbook by using the Excel Tools Share Workbook dialogue.
- In an open shared workbook the contents are not updated until you have done a file save in Excel.
- Writing to a shared workbook can be painfully slow.
- In general, it is best to close the workbook.

6.29.6.3 Writing Parameter to Spreadsheet

At first, we create a GDX file, containing some random data:

* file makeData.gms

Set
  i / i1*i4 /
  j / j1*j4 /
  k / k1*k4 /;

Parameter V(i,j,k);
V(i,j,k)$(uniform(0,1) < 0.30) = uniform(0,1);
When we run this GAMS model from the command prompt using the following statement, the file writingPar.gdx will be created at the end of the run.

GAMS makeData gdx=writingPar

Using the file writingPar.gdx, we can write to a spreadsheet:

Write parameter V to the first cell in the first sheet; because we only specify the top left corner of the sheet, the complete sheet can be used to store the data. We do not specify the row and column dimension, so they will be set to rDim=2 and cDim=1 by default. (See also dimensions)

By using the following command (remember to close an already existing file writingPar.xlsx in advance or make it a shared notebook as discussed in the previous example):

gdxxrw writingPar.gdx output=writingPar.xlsx par=V rng=A1

The steps above can be combined in a single GAMS model using the execute_unload and execute statements as follows:

Set
   i / i1*i4 /
   j / j1*j4 /
   k / k1*k4 /;
Parameter V(i,j,k);
V(i,j,k)$(uniform(0,1) < 0.30) = uniform(0,1);
execute_unload 'writingPar.gdx', i, j, k, V;
execute 'gdxxrw writingPar.gdx par=V rng=A1';

The resultant spreadsheet looks like:

Note that if we only want to write the parameter V, there is no need to unload the sets i, j and k explicitly. The labels written to the columns A and B and to the first row are stored directly together with V in the GDX file.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample11b] for reference.
6.29.6.4 Writing to Spreadsheet with merge Option

When writing to a spreadsheet one can control data handling and matching using the `merge` command line option. When `merge` is active, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already. Also under `merge`, spreadsheet cells for which there is no matching row/column pair will not be changed. This option might be very useful, e.g., if there is a fixed report layout/framework already in your spreadsheet file which should not be changed when writing the data from GAMS.

Suppose we want to write the parameter $A$, already stored in data.gdx, to an existing spreadsheet:

In the following spreadsheet, there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non existing row labels 'san francisco.chicago':

Use the following commands to write the data from GDX to spreadsheet twice to different ranges for comparison:

```
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B1:G4 rDim=2 cDim=1 merge
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B8     rDim=2 cDim=1
```

Note that `merge` is a symbol option affecting only the symbol $A$. The resultant spreadsheet looks like:

The parameter is written to the range B8-F12 without `merge` enabled, while the option is enabled when writing to the range B1:G4, respecting the data arrangement already existing. Note that the column and row orders vary and the san francisco - chicago row is missing since it is not mentioned in the labels within the spreadsheet before the merge operation, while the horse column is still present with it's data left alone, not being overwritten by the parameter.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample12] for reference.

Note

- Using the `merge` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `merge` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements at all.
- The matching of labels is not case-sensitive.
- Warning: Enabling the `merge` option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.
6.29.6.5 Writing to Spreadsheet with clear Option

When writing to a spreadsheet one can also use the `clear` option to control data handling and matching. When `clear` is enabled, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already but all data and formulas in the target range will be removed.

Suppose the parameter A from the previous example is stored in data.gdx and there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non-existing row labels 'san francisco.chicago':

Use the following command to write to exampleData.xlsx with `clear` enabled:

```bash
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!I1 rDim=2 cDim=1 clear
```

then the result is

The results are similar to those under merge but the old data in the column labeled 'horse' has been removed.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample12] for reference.

Note

- Using the `clear` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `clear` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements.
- The matching of labels is not case-sensitive.
- Warning: The `clear` option will clear all Excel formulas and values in the rectangle used, even if the cells do not have matching row/column labels in the GDX file.

6.29.6.6 Writing to Spreadsheet using a Filter

In Excel, you can filter the data by some specified conditions so that only the data matching the conditions is displayed. This might be useful in some cases, e.g. it helps you to focus on the most relevant data within a large table of data. With GDXXRW you can add some basic filter to your spreadsheet when writing data from a GDX file.

The following example creates a small GDX file with some random data, which is used to write the symbol A to a spreadsheet later on with the `filter` option enabled.

```bash
Set
  i / i1*i2 /
  j / j1*j2 /
  k / k1*k2 /;

Parameter A(i,j,k);
A(i,j,k) = uniform(0,1);
execute_unload 'test.gdx', A;
execute 'gdxxrw test.gdx filter=1 par=A rDim=1 cDim=2 rng=Sheet1!A1';
```
Since \texttt{filter} is a global option, it must be specified in advance of the symbols for which you want to add a filter. The default value is zero, i.e. no filter will be added. If there are multiple rows in the column header, i.e. \texttt{cDim} is greater than zero, the valid range for the filter option is \texttt{1..cDim}.

The screenshot above shows the filter in Excel. When we specify \texttt{filter=2} in this example with two dimensions for the column header, the row with the filter moves away from the data range as illustrated below:

One could now filter the data, e.g. displaying only the values where the label of the first dimension is \texttt{i1} by selecting this value exclusively within the drop down menu of column A.

This example is also part of the GAMS Data Utilities Library, see model \[GDXXRWExample15\] for reference.

### 6.29.6.7 Writing to Spreadsheet adding Text and Hyperlinks

The following example illustrates the use of the \texttt{text directive}. Adding text and hyperlinks to your spreadsheet is useful to customize the output and to navigate more quickly through the data.

First, we write some data to a GDX file and we use the text directive to write text to various cells in the spreadsheet; some of the cells are hyperlinks to other locations. To increase readability, we’ll use a parameter file \texttt{howToWrite.txt} to shorten the \texttt{GDXXRW} statement.

```plaintext
Set
  i / i1*i9 /
  j / j1*j9 /;

Parameter A(i,j);
A(i,j) = 10*ord(i) + ord(j);
execute_unload 'pv.gdx' A;

$onEcho > howToWrite.txt
  text="Link to data" rng=Index!A2 linkID=A
  text="Below the data for symbol A" rng=data!C2
  par=A rng=data!C4
  text="Back to index" rng=data!A1 link=Index!A1
  text="For more information visit GAMS" rng=data!C1 link=http://www.gams.com
$offEcho

execute 'gdxxrw pv.gdx output=pv.xlsx @howToWrite.txt'
```

We will write the text "Link to data" to the cell A2 of sheet Index. The option \texttt{linkID} is used to add a hyperlink to the range of the symbol \texttt{A}. In addition, we create a hyperlink "Back to index" in the cell A1 of sheet data using the \texttt{link} option pointing to the cell A1 of sheet Index. One can also specify links to external sources. For demonstration, we add a link to the GAMS homepage.

Below a screen shot showing both sheets data and Index created by the commands above:

This example is also part of the GAMS Data Utilities Library, see model \[GDXXRWExample16\] for reference.
6.29.7 Reading and Writing, Extended Use - Examples:

6.29.7.1 Reads a Table from Spreadsheet, manipulates the Data and writes back to Spreadsheet

In the following example, we read data from a spreadsheet and save the data in a GDX file. Using the \$gdxIn and $load GAMS directives, we load the data from the GDX file into GAMS afterwards. The GAMS program modifies the data and at the end of the run the data is saved in a new GDX file (tmp.gdx). The last step updates the spreadsheet with the modified parameter.

We read the spreadsheet and load the data from the resultant GDX file at compile time. The data modification and the unloading and writing process are done at execution time, using the execute_unload and execute directives:

$call gdxxrw test1.xlsx dSet=I rng=A2:A3 rDim=1 dSet=A rng=B1:D1 cDim=1 par=X rng=A1:D3 rDim=1 cDim=1
$gdxIn test1.gdx
Set I(*), A(*);
$load I A
Parameter X(I,A);
$load X
display I, A, X;
$gdxIn
X(I,A) = - X(I,A);
execute_unload 'tmp.gdx', I, A, X;
execute 'gdxxrw tmp.gdx output=test1.xlsx par=X rng=EX6!A1:D3 rDim=1 cDim=1';

The parameter is written to the sheet EX6. However, since we only write the parameter X, we do not necessarily have to unload the set I and A.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample13] for reference.

6.29.7.2 Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve

In this example, we use a modified version of the [trnsport] model from the GAMS model library to demonstrate the process of reading data, defining and solving the model and writing a solution report to spreadsheet altogether. This example illustrates in particular:

- Compilation phase
- Read data from a spreadsheet and create a GDX file
- Loading sets from the GDX file
- Using the sets as a domain for additional declarations
- Reading additional data elements
• Execution phase
• Solve the model
• Write solution to a GDX file
• Use GDX file to update spreadsheet

A parameter file howToRead.txt is used to increase the readability of the GDXXRW call. Note that the dimension of the scalar we want to read is set to zero. The range of each parameter can be specified by using the top left corner only, since there are two empty rows separating the data blocks from each other and the default value of skipEmpty signals to stop reading if two empty rows occur.

```plaintext
$onEcho > howToRead.txt
dSet=i rng=A3:A4 rDim=1
dSet=j rng=B2:D2 cDim=1
par =d rng=A2 rDim=1 cDim=1
par =a rng=A8 rDim=1
par =b rng=A13 rDim=1
par =f rng=A19 dim=0
$offEcho
$call gdxxrw TrnsportData.xlsx @howToRead.txt
$gdxIn TrnsportData.gdx

Set
  i(*) 'canning plants'
  j(*) 'markets';
$load i j
display i, j;

Parameter
  a(i)  'capacity of plant i in cases'
  b(j)  'demand at market j in cases'
  d(i,j) 'distance in thousands of miles';
Scalar f 'freight in dollars per case per thousand miles';
$load d a b f
$gdxIn

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
  x(i,j) 'shipment quantities in cases'
  z    'total transportation costs in thousands of dollars';
Positive Variable x;

Equation
  cost   'define objective function'
  supply(i) 'observe supply limit at plant i'
  demand(j) 'satisfy demand at market j';
cost..   z =e= sum((i,j), c(i,j)*x(i,j));
supply(i).. sum(j, x(i,j)) =l= a(i);
demand(j).. sum(i, x(i,j)) =g= b(j);
```
Model transport / all /

solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'TrnsportData.gdx', x;
execute 'gdxxrw TrnsportData.gdx output=TrnsportData.xlsx squeeze=n var=x.l rng=Sheet2!A1';

The solution is written to Sheet2 of the input file TrnsportData.xlsx by executing GDXXRW at execution time. The var statement is used in the symbol specification to write out the level of variable x. In order to write zero values, the squeeze option is disabled. Otherwise, the cells C4 and D3 remain blank.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample14] for reference.

6.29.7.3 Reading Special Values from Spreadsheet and writing to Spreadsheet

This example demonstrates the reading and writing of special values with GDXXRW.

Reading Special Values
Assuming we want to read the special values stored in the spreadsheet below:

The following statement reads parameter vIN from the spreadsheet above:

gdxxrw exampleData.xlsx output=specialValues.gdx NaIn=N/A squeeze=n par=vIN cDim=1 rng=specialValues!A1:S2

To affect the parameter vIN, the global options NaIn and squeeze (to display zero values in the GDX file) must be specified in advance.

The cells containing the strings Eps, +Inf, -Inf, Inf, NA and Undf are read in correctly. The division by zero error in the spreadsheet will be written as Undf. We defined a new string - 'N/A' - within the option NaIn to be recognized as NA additionally. Within GAMS, there is no directly comparable data type for 'None' and 'Null', so these strings will be interpreted as Undf. Note the importance of the cell format specified within Excel. For instance, the values for v13 and v14 are different, although both fields contain a Dollar sign, since the cell format of M2 is 'General', while the cell format of N2 is 'Currency'. The boolean 'False' turns into zero after reading, while 'True' turns into -1. Note here, that this is independent whether the booleans are written as plain text within Excel or by using the formulas '='True()' resp. '='False()' !. We disabled squeeze in order to display the zero values for the elements v16, v18 and v19 in the GDX file.

Writing Special Values
Initially, we declare a parameter with special values and generate a GDX file from it:

$onUndf
Set v / v1*v7 /;
Parameter vOUT(v) / v1 Eps, v2 +Inf, v3 -Inf, v4 Inf, v5 Undf, v6 0.0, v7 NA /;
Scalar S / 0 /;

execute_unload 'specialValues.gdx', v, vOUT, S;
We will write the parameter \texttt{vOUT} to spreadsheet twice. At first to demonstrate the usage of \texttt{EpsOut, plnInfOut, mInfOut, UndfOut, zeroOut} and \texttt{NaOut}, the second time to show the default settings when writing. Those output string options may be useful, if the GAMS default settings are not appropriate for your Excel calculations later on or to customize the representation of the values in Excel in general. Note that there is also a scalar declaration in order to demonstrate the different behavior when writing scalars and parameter with zero values to Excel while using the \texttt{zeroOut} option.

\$onEcho > howToWrite.txt
* defining new strings to be used when writing special values:
\texttt{EpsOut=0 plnInfOut=+1E+100 mInfOut=-1E+100 UndfOut=undefined zeroOut=zero NaOut=notAvailable}

* now write parameter \texttt{vOUT} with merge to force the column F containing set element "v6" and \texttt{vOUT("v6")}
\texttt{set=v \ rang=specialValues!A6:G6 cDim=1}
\texttt{par=vOUT \ rang=specialValues!A6:G7 cDim=1 merge}
\texttt{text="Special values of Parameter \texttt{vOUT} written with user defined output strings:" \ rang=specialValues!A5}
\texttt{text="Scalar S / 0 /:" \ rang=specialValues!I6}
\texttt{par=S \ rang=specialValues!I7}

* reset the strings for special values back to default und write \texttt{vOUT} again
\texttt{resetOut}
\texttt{set=v \ rang=specialValues!A10:G10 cDim=1}
\texttt{par=vOUT \ rang=specialValues!A10:G11 cDim=1 merge}
\texttt{text="Special values of Parameter \texttt{vOUT} written with default output strings:" \ rang=specialValues!A9}
\texttt{text="Scalar S / 0 /:" \ rang=specialValues!I10}
\texttt{par=S \ rang=specialValues!I11}
\$offEcho

eexecute 'gdxxrw specialValues.gdx output=exampleData.xlsx @howToWrite.txt';

In order to increase readability when executing GDXXRW, we use a parameter file named howToWrite.txt and additionally, we write some text out to structure the Excel file. The range for the non-default values is A6:G7. We define the new strings to be used for the special values first, affecting the following symbols. To write the default values to the range A10:G11, we use the \texttt{resetOut} option to reset the output strings to default, otherwise, the new strings remain in effect, since they are global options.

As mentioned briefly above, the \texttt{zeroOut} option affects the scalar \texttt{S} and the parameter \texttt{vOUT} differently. While we get the expected result - 'zero' - for the scalar \texttt{S}, the cell F7 for the zero value of \texttt{vOUT} remains empty, since zero values of parameters are not even part of the GDX file in general (and cannot be added from your GAMS model; Note here, that the \texttt{squeeze} options only affects the writing of sub-fields of variables and equations). Therefore, cell F11 has no value, too. Though scalars with zero values are stored in the GDX file. We'll present a workaround for writing zero values of a parameter to spreadsheet in the next example Writing Parameter to Spreadsheet including Zero Values.

The complete example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample8] for reference.

\textbf{Note}

\begin{itemize}
\item When writing to a spreadsheet, special values such as \texttt{Eps}, \texttt{NA}, \texttt{Undf} and \texttt{Inf} will be written but this can be changed. When reading data from a spreadsheet, the ASCII strings for these special character strings will be used to write corresponding special values to the GDX file.
\item Cells that are empty or zero will not be written to the GDX file.
\end{itemize}
6.29.7.4 Writing Parameter to Spreadsheet including Zero Values

There is no straight way to write zero values of a GAMS parameter to spreadsheet from your model using GDXXRW, since zero values of a parameter are not stored in the GDX file (see also the previous example Reading Special Values from Spreadsheet and writing to Spreadsheet). However, instead of the zero values one can store an EPS in the GDX file and instruct GDXXRW to use a zero when writing the value for EPS using the epsOut option afterwards as demonstrated below:

```gams
Set i / i1*i9 /;
Parameter A(i), Amod(i);
A(i) = uniformInt(0,1);
* Amod(i) = A(i) if A(i) <> 0 and Amod(i) = EPS if A(i) = 0
Amod(i) = EPS$(not A(i)) + A(i);
* Unload the unmodified and modified parameter and write to spreadsheet using an option file
execute_unload 'zeroPar1.gdx' A Amod;
```

This approach is impracticable in the unlikely event that there are already EPS values in your parameter and you want to write these as EPS to your spreadsheet.

An alternate approach regarding variables while exploiting the squeeze option is demonstrated below (parameter A and set i refer to the data above):

```gams
Variable dummyPar(i);
dummyPar.l(i) = A(i);
* In order to write every entry of dummyPar in the spreadsheet, one must allocate
* a non-zero value to one of the variable attributes .m, .lo or .up
dummyPar.up(i) = 1;
* Unload the dummy variable and write the .l subfield to spreadsheet while disabling squeeze
execute_unload 'zeroPar2.gdx' dummyPar;
```

Creating the additional variable and the allocation of a non-zero value to one of the other variable attributes are the drawbacks of this approach. Note that we only declared the parameter Amod to keep the original data of A untouched in order to run the code in a single model and to write A to spreadsheet, too.

This example is also part of the GAMS Data Utilities Library, see model [GDXXRWExample19] for reference.
6.29.7.5 Reading several Scalars from Spreadsheet

Suppose you want to read a large number of scalars (i.e. scalar names and their associated values) for your model from a spreadsheet file as shown below. Originally, the data was taken from the model \text{[indus89]} of the GAMS Model Library.

Natively, one could declare all scalars directly inside the model and read every single one of them with \text{GDXXRW} afterwards (the following code is shortened for representation):

Scalar baseyear, repco;
$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx par=baseyear rng=indus89Scalars!B2 dim=0 par repco rng=indus89Scalars!B3 dim=0 trace=0
$gdxIn indus89Scalars.gdx
$load baseyear repco

However, the code would become quite long and circumstantially for this example, especially the \text{GDXXRW} statement, because every scalar must be specified individually. Therefore, we present a more sneaky way to tackle the problem. At first, the file mkScalar.gms is written. We declare a set and a parameter to hold the scalar names and their values. A simple \text{GDXXRW} call is used to read the names and values from the file exampleData.xlsx. Afterwards, we load the data using $\text{gdxIn}$ and $\text{load}$. The result of this is a single parameter containing the scalar values defined over a set containing the scalar names. However, this is not exactly what we are looking for. To transform this representation into several single scalar definitions, we use a simple loop over the set \text{scalarNames} and the \text{Put Writing Facility} to generate a file scalars.gms, containing those single scalar statements.

$\text{onEchoV > mkScalar.gms}
Set scalarNames;
Parameter scalarValues(scalarNames);
$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx set=scalarNames rng=indus89Scalars!A2:A22 rDim=1 par=scalarValues rng=indus89Scalars!A2:B22 rDim=1
$ifE errorLevel<>0 $abort Problems reading sheet indus89Scalars with GDXXRW
$gdxIn indus89Scalars.gdx
$load scalarNames scalarValues
File fs / 'scalars.gms' /;
put fs;
loop(scalarNames, put / 'Scalar ' scalarNames.tl ' / ' scalarValues(scalarNames):>20:10 ' /';');
$\text{offEcho}
$call gams mkScalar
$include scalars.gms

Calling the file mkScalar.gms will create the file scalars.gms, containing all those scalar statements (shortened for representation):

Scalar baseyear / 1988.0000000000000000 /;
Scalar repco / 2.5000000000000000000 /;

Finally, the file scalars.gms is included to your model.

This example is also part of the GAMS Data Utilities Library, see model \text{[GDXXRWExample17]} for reference.
6.29.8 Changes in the Set Values Parameter

The following document some changes that were made when reading a set using the `values=string` option. Reading a domain or a parameter was not affected by these changes.

To illustrate the various behaviors in different versions of GDXXRW, we are using the spreadsheet data as shown below, using the following call:

```
$call gdxxrw test.xlsx set=one rng=B2 rDim=1 values=string set=two rng=B1 rDim=1 cDim=1 values=string
```

We read the one dimensional set in column B by specifying the top-left corner of the data (cell B2) or the full range (B2..B5). A two dimensional set is read using the top-left corner of the data (cell B1) or the full range (B1..E5). Variations are introduced by specifying options for `values` to be `string` or `yn` or `noData`. The option value `all` is only available in later versions of GDXXRW and was used to introduce the same behavior as `strings` in earlier versions.

6.29.8.1 Chronological description of the changes made to the Values option:

- **GAMS versions prior to version 24.3:**
  
  `values=string` results in reading the data dense. The contents of a cell is used for the set associated text and an element is included even if the data cell is empty.

- **GAMS version 24.3.1**
  
  We changed the interpretation of `string` to mean that the set element was only to be included when the string data was not empty. Note below that element `a4` is missing from the one-dimensional set and so are `a1.b1`, `a2.b2` etc from the two-dimensional set. For the two-dimensional case this looked more or less how the GAMS compiler interprets a table statement. Unfortunately, the interpretation of empty data cells was also applied to one-dimensional sets leading to undesired results.

- **GAMS version 24.4.1**
  
  Recognizing that reading a set dense was no longer available, we introduced a new option `values=all`. This allowed us to read Excel data the same way as was possible before version 24.3.1 using the `values=string` i.e. reading the data dense and including all cells whether the data cell is empty or not and use the content of the data cell for the set associated texts.

- **GAMS version 24.4.6 (Current status)**
  
  We decided to remove some confusion with the interpretation of the `values` option by introducing the options `dense` and `sparse` and flagging the options `strings` and `all` as deprecated. Both `strings` and `all` are replaced with `dense`.

Backward compatibility issues.

With these changes we broke our in house rule not to introduce changes that break backward compatibility. Because of this, the user needs to change the parameters for the GDXXRW call or change the workbook data. The parameters for the call should be changed from `values=string` to `values=dense` for one-dimensional sets where we specify the top-left corner only. In case the data in the workbook needs to be changed, inserting a string to in the data cell will address the issue. In the example on the top of the page, inserting a 'Y' in cell C5.
6.30 INVERT

INVERT inverts a matrix.

6.30.1 Usage

invert gdxin i a gdxout inva

where

gdxin
   name of gdxfile with matrix
i
   name of set used in matrix
a
   name of 2 dimensional parameter inside gdxin
gdxout
   name of gdxfile for results (inverse matrix)
inva
   name of 2 dimensional parameter inside gdxout

Calculates the inverse of a matrix $a(i,j)$, where $i$ and $j$ are aliased sets. The matrix inva in gdxout will contain the inverse.

6.30.2 Example

$onText
Finds the inverse of a matrix through an external program.

Erwin Kalvelagen, march 2005

Reference: model gauss.gms from the model library
   http://www.gams.com/modlib/libhtml/gauss.htm
$offText

Set i /i1*i3 /
Alias (i,j);

Table a(i,j) 'original matrix'
   i1  i2  i3
   i1  1   2   3
   i2  1   3   4
   i3  1   4   3;

Parameter inva(i,j) 'inverse of a';

execute_unload 'a.gdx', i, a;
execute '=invert.exe a.gdx i a b.gdx inva';
execute_load 'b.gdx', inva;

display a, inva;

$onText
### 6.31 MCFilter

**MCFilter** filters duplicate and dominated points from a solution set.

#### 6.31.1 Usage

`mcfilter InputFile`

The *input* is a gdx file with the following data:

- **parameter X(point, i)**
  
  Points containing binary values; use EPS when all zero for a point

- **parameter F(point, obj)**

  Objectives for the points X; use EPS when all zero for a point

- **parameter S(obj)**

  Direction of each objective: 1=max, -1=min

The *output* will be a gdx file called `InputFile_res.gdx` with the same parameters but without duplicates and dominated points.

#### 6.31.2 Example

* Generate random data to test MCFilter

```plaintext
Set
    Ri / R1*R65535 /
    Xi / X1*X16 /
    Fi / F1*F5 /
;
Parameter
    X(Ri,Xi) 'the binary variables'
    F(Ri,Fi) 'calculated F variables'
    S(Fi) 'sign on the F variables';

* The following points will be unique

X(Ri,Xi) = mod(floor(Ri.ord/power(2, Xi.off)), 2);
F(Ri,Fi) = uniform(1,2);
S(Fi) = -1 + 2$(uniform(0,1) <= 0.5);
execute_unload 'testdata.gdx', X F S;
exectue 'mcfilter testdata.gdx';

* The file testdata_res.gdx will contain the results
```

Below is the partial log after running the above model:

```
mcfilter v3.
Number of records       = 65535
Number of X variables   = 16
Number of F variables   = 5
Loading GDX data        = 156 ms
After X Filter, count   = 65535
X Duplicate filter      = 16 ms
After F Filter, count   = 1070
F Dominance filter      = 249 ms
Writing GDX data        = 16 ms
```
6.32 MDB2GMS

Author
Erwin Kalvelagen

6.32.1 Overview

MDB2GMS is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (.mdb or .accdb) and the target is a GAMS Include File (.inc) or a GAMS GDX File (.gdx).

When running the executable mdb2gms.exe without command-line arguments, the tool will run interactively with a built-in GUI interface. Alternatively MDB2GMS can be run in batch mode, which is useful when running it directly from a GAMS model without user intervention using the $call command at compile time or the execute command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

6.32.2 Requirements

MDB2GMS runs only on Windows PC's and with MS Access installed. MS Access comes with certain versions of MS Office, but some MS Office versions will not include Access. The actual retrieval of the database records is performed by DAO or Data Access Objects, an object layer for accessing the database. The actual database is the Jet engine, which performs the queries and retrieves the data. Also consider to use SQL2GMS instead of MDB2GMS, if MS Access is not installed on your system.

To use this tool effectively you will need to have a working knowledge of SQL in order to formulate proper database queries.

6.32.3 Batch Usage

MDB2GMS can be run in batch mode without user intervention from within the GAMS model by using the $call resp. execute statements or directly from command prompt while specifying all arguments in the command-line. A MDB2GMS batch call is of the following form:

`mdb2gms inputFiel outputFile queryString`

A proper batch call will at least contain the following three command-line arguments:
1. The name of the MS Access database `inputFile` must be specified (.mdb or .accbd format). Use the `I` argument to enter the file name, i.e. `I=inputFile`.

2. The name of `outputFile`, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option `O`, i.e. `O=outputFile.inc`, while the use of a GDX file is indicated by the option `X`, i.e. `X=outputFile.gdx`.

3. The SQL `queryString`, containing the SQL statement to be executed on the database, must be specified within the option `Q`, i.e. `Q=queryString`.

See also Command-line Arguments below for a complete list of all possible command-line arguments. Consider that the `$call` or `execute` usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the interactive built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a `@` (At sign).

Also consider to take a look at the section Strategies, mentioning some of the drawbacks of the batch usage and how to overcome them.

If you only specify `I=inputFile` then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command-line argument. Only if an input file, an output file and a query string is provided, the call will be considered as batch invocation.

### 6.32.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using `MDB2GMS` directly from the GAMS model or command prompt.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Interpretation</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>inputFile</td>
<td>none</td>
<td>Specify the name of the input file (required).</td>
</tr>
<tr>
<td>O</td>
<td>outputIncludeFile</td>
<td>none</td>
<td>Specify the name of the output file (.inc). Either <code>O=</code> or <code>X=</code> must be specified (or both).</td>
</tr>
<tr>
<td>On</td>
<td>n-th outputIncludeFile</td>
<td>none</td>
<td>Match the nth query with the nth output file (.inc format) if multiple queries are used.</td>
</tr>
<tr>
<td>X</td>
<td>outputGDXFile</td>
<td>none</td>
<td>Specify the name of the output file (.gdx). Either <code>O=</code> or <code>X=</code> must be specified (or both).</td>
</tr>
<tr>
<td>Q</td>
<td>Query</td>
<td>none</td>
<td>This option can be used to specify a SQL query (required).</td>
</tr>
<tr>
<td>Qn</td>
<td>n-th Query</td>
<td>none</td>
<td>Match the nth query with the nth output file (.inc format) or with the nth set- or parameter name when writing to GDX if multiple queries are used.</td>
</tr>
<tr>
<td>S</td>
<td>setName</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.</td>
</tr>
<tr>
<td>Sn</td>
<td>n-th setName</td>
<td>none</td>
<td>Match the nth query with the nth set in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>Argument</td>
<td>Interpretation</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Y</td>
<td>setName (with expl. text)</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.</td>
</tr>
<tr>
<td>Yn</td>
<td>n-th setName (with expl. text)</td>
<td>none</td>
<td>Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>P</td>
<td>parameterName</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.</td>
</tr>
<tr>
<td>Pn</td>
<td>n-th parameterName</td>
<td>none</td>
<td>Match the nth query with the nth parameter in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>D</td>
<td>Debug</td>
<td>disabled</td>
<td>Generate debug information.</td>
</tr>
<tr>
<td>B</td>
<td>Quote Blanks</td>
<td>disabled</td>
<td>Quote strings if they contain blanks or embedded quotes.</td>
</tr>
<tr>
<td>M</td>
<td>Mute</td>
<td>disabled</td>
<td>Controls if additional information is written to the log and include file.</td>
</tr>
<tr>
<td>L</td>
<td>Listing</td>
<td>disabled</td>
<td>Controls if the data is embedded in the listing file.</td>
</tr>
<tr>
<td>@fileName</td>
<td>ext. options file</td>
<td>none</td>
<td>Causes the program to read options from an external text file.</td>
</tr>
<tr>
<td>N</td>
<td>iniFileName</td>
<td>mdb2gms.ini</td>
<td>Indicates the usage of a different INI file.</td>
</tr>
<tr>
<td>F</td>
<td>formatString</td>
<td>none</td>
<td>Specify a format string.</td>
</tr>
<tr>
<td>W</td>
<td>Wiring</td>
<td>none</td>
<td>Maps database columns to GAMS index positions.</td>
</tr>
<tr>
<td>R</td>
<td>rowBatchSize</td>
<td>100</td>
<td>Row batch size; the default is 100 records.</td>
</tr>
</tbody>
</table>

Some more detailed remarks on the command-line arguments:

\[ I = \text{string} \text{ (inputFile, default = none)} \]

This option is required and specifies the name of the .mdb or .acpdb file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. On a network UNC Names can be used, and files from another computer can be accessed, e.g.

"\hostname\c\my documents\a.mdb."

This option is required for batch processing. To specify a path equal to the location where the .gms file is located, you can use:

I=system.fpmydb.mdb

This option is demonstrated in most examples, see Example 1 - Reading a single valued Table for instance.
This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is writable. UNC Names can be used. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the $include command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in most examples, see Example 1 - Reading a single valued Table for instance.

When using multiple queries in a single MDB2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

```plaintext
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC names can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both). This option is demonstrated in Example 5 - Reading Set with Explanatory Text or Example 7 - Multi-Query Batch Example for instance.

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by Access we refer to the documentation that comes with MS Access. The query is passed on directly to the Jet database engine, so the complete power and expressiveness of Access SQL is available. For an exact description of allowed expressions consult a text on MS Access.

One notable syntax feature is that when field names or table names contain blanks, they can be specified in square brackets. Examples:

```plaintext
Q="SELECT * FROM mytable"
Q="SELECT year, production FROM [production table]"
Q="SELECT [GAMS City], value FROM [example table], CityMapper WHERE [Access City]=city"
```

This option is demonstrated in Example 1 - Reading a single valued Table for instance.
When using multiple queries in a single MDB2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX.

Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc

or (GDX output file format - where several sets and parameters can be stored in a single file):

Q1="SELECT a, b FROM table"
P1=abParameter
Q2="SELECT c FROM table"
S2=cSet

Note the usage of the arguments Pn resp. Sn in order to store the results as parameter resp. set and to specify the name of the symbols. See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

S = string (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in Example 4 - Reading a multi dimensional Set.

Sn = string (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

Y = string (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in Example 5 - Reading Sets with Explanatory Text for instance.

Yn = string (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.
Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet

The last column specified within the select clause in the SQL statements will be
used as explanatory text.

\[ P = \text{string (parameterName, default = none)} \]

If we write to a GDX file, use this option to specify the name of a parameter to be
stored the GDX file (containing the results of the query).

\[ P_n = \text{string (parameterName, default = none)} \]

If multiple queries are used in a single \texttt{MDB2GMS} call while writing to a GDX file, use
this option to specify the name of the nth parameter to be stored in the GDX file
(containing the results of the nth query), e.g.

\[
\begin{align*}
Q1="SELECT i, j, value FROM table" \\
A1=ijValue \\
Q2="SELECT n, m, value FROM table" \\
A2=nmValue
\end{align*}
\]

See also section \texttt{Multi-Query Batch Usage} or \texttt{Example 7 - Multi-Query Batch Example}
for instance.

D (Debug, default = disabled)

This option can be used for debugging purposes. If specified the import filter will
not run minimized but "restored", i.e. as a normal window. In addition the program
will not terminate until the user clicks the Close button. This allows you to monitor
possible errors during execution of \texttt{MDB2GMS}.

B (Quote Blanks, default = disabled)

If this parameter is specified, strings that have blanks in them will be quoted. If
the string is already quoted this step is not performed. If the name contains an
embedded single quote, the surrounding quotes will be double quotes. If the name
already contains a double quote, the surrounding quotes will be single quotes. If
both single and double quotes are present in the string, then all double quotes are
replaced by single quotes and the surrounding quotes will be double quotes. By
default this option is turned off. For more information see subsection \texttt{Quotes}. This
option only applies to an output include file.

M (Mute, default = disabled)

Run in modest or mute mode: no additional information, such as version number,
number of rows in the data, elapsed time, used query etc. is written to the log and
include file.

L (Listing, default = disabled)

Embed the data between the \$offListing and \$onListing dollar control options,
so the data will not be listed in the listing file. This is a quick way to reduce the
size of the listing file when including very large data files into the model. Otherwise
the listing file would become too large to be handled comfortably.

\[ @\text{fileName} = \text{string (fileName, default = none)} \]
Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also Command Files for some further details.

\[ \text{N} = \text{string} \text{ (fileName, default = mdb2gms.ini)} \]

Use a different INI file than the standard mdb2gms.ini located in the same directory as the executable mdb2gms.exe.

\[ \text{F} = \text{string} \text{ (formatString, default = none)} \]

In special cases we can apply a format string on the include file output (not for GDX output). Each column in the result set is a string and can be represented by a \( s \) in the format string.

\[ \text{W} = \text{string} \text{ (wiring, default = none)} \]

By using the \( W \) option, one can map database columns to GAMS index positions. See model [Wiring] for reference.

\[ \text{R} = \text{integer} \text{ (rowBatchSize, default = 100)} \]

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of MDB2GMS.

6.32.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the [trusport] model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.mdb).

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM   distances
```

By running the following MDB2GMS statement, the connection to the database Sample.mdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
```
The MS Access database file name is specified using the argument I. Note that the string is enclosed by quotes, as the string contains blanks. The arguments Q and O are used to specify the query and the output file name (and format).

The generated include file distances.inc looks like:

```plaintext
* -----------------------------------------------------
* MDB2GMS 24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 14.0
* Jet version: 4.0
* Database:  F:\datalib\Sample.mdb
* Query: SELECT city1, city2, distance FROM distances
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```

The commented header section summarizes some information about the MDB2GMS resp. GAMS version and about the executed database query. The standard export format is to consider the last column as the value column (containing the distances) and the previous columns as the indices (containing the city names). The indices are separated by a dot, allowing the generated include file to be used as part of a parameter declaration statement in your GAMS model.

Retrieving the data using MDB2GMS from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

Set
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /;

$call mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc
/

display d;

Finally, the values of the parameter d are displayed:

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.500</td>
<td>1.700</td>
<td>1.800</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.500</td>
<td>1.800</td>
<td>1.400</td>
</tr>
</tbody>
</table>

This example is also part of the GAMS Data Utilities Library, see model [Distances1] for reference. Note that the query results are written to a GDX file in addition.
6.32.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want to read the three index columns year, loc, prod and the value columns sales and profit from the database file Sample.mdb. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for sales resp. profit) and a UNION select.

Consider the table with two value columns sales and profit:

![Table: data (shortened for presentation)](image)

**Two separate Parameters**

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year, loc, prod, sales
FROM data
SELECT year, loc, prod, profit
FROM data
```

We can generate a include file sales.inc by running the following command:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
```

Note that we specify the first query in order to select the sales and the relevant index columns within the Q argument. The query results are written to sales.inc using the O argument. Analogously we generate a include file profit.inc by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
```

Retrieving the data using MDB2GMS from the database Sample.mdb and including the queried data in your GAMS model within the parameter declaration statements of sales and profit (at compile time) can be combined in the following way:

```
Set
  year 'years' / 1997*1998 /
  loc 'locations' / nyc, was, la, sfo /
  prd 'products' / hardware, software /;
$call mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
Parameter sales(year,loc,prd) /
```
$include sales.inc
/

$call mdb2gms I=Sample.mdb" Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
Parameter profit(year,loc,prd) /
$include profit.inc
/

This example is also part of the GAMS Data Utilities Library, see model [SalesProfitDB1] for reference.

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index type to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

SELECT year, loc, prod, 'sales', sales FROM data
UNION
SELECT year, loc, prod, 'profit', profit FROM data

The data is accessed, queried and written to data.inc by running the following command:

mdb2gms @howToRead.txt

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc

The generated include file data.inc looks like (shortened for presentation):

* -----------------------------------------------------
* MDB2GMS 24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 14.0
* Jet version: 4.0
* Database: F:\datalib\Sample.mdb
* Query: SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
* -----------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
Retrieving the data using **MDB2GMS** from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index **type** in order to access the data type **sales** resp. **profit**):

```gams
$onEcho > howToRead.txt
I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc
$offEcho

Set
  year 'years' / 1997*1998 /
  loc 'locations' / nyc, was, la, sfo /
  prd 'products' / hardware, software /
  type 'data type' / sales, profit /;

$call mdb2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
$include data.inc
/;
```

This example is also part of the GAMS Data Utilities Library, see model [SalesProfitDB2c](#) for reference.

### 6.32.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a MS Access database file. Suppose we want to read the column City1 (see table distances) in order to define the set \( i \) in the GAMS model. Make sure elements are unique by using the distinct operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```sql
SELECT distinct(City1)
FROM distances
```

The include file city1.inc looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
SAN-DIEGO
SEATTLE
* -----------------------------------------------------
```

All steps (data access via **MDB2GMS**, set definition) can be combined:
$call mdb2gms I=Sample.mdb Q="SELECT distinct(city1) FROM distances" O=city_i.inc
Set i 'canning plants' /
$include city_i.inc
/

display i;

The display statement generates the following output in the listing file:

---- 56 SET i
seattle , san-diego

6.32.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the O argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing a b c is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. a.b.c will be interpreted correctly.

There are different ways to add these periods explicitly within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (index1, index2, index3 and dataTable are some placeholders):

```
SELECT index1, index2, index3, '' FROM dataTable
```

or by adding the periods explicitly within the select clause (|| or & depending on DBMS):

```
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table distances stored in Sample.mdb. The following MDB2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, '' FROM distances" O=city_ij.inc
```

The include file city_ij.inc looks like (header informations are removed in order to shorten the representation):
*-------------------------------------------------------
SEATTLE.NEW-YORK ''
SAN-DIEGO.NEW-YORK ''
SEATTLE.CHICAGO ''
SAN-DIEGO.CHICAGO ''
SEATTLE.TOPEKA ''
SAN-DIEGO.TOPEKA ''
*-------------------------------------------------------

Without adding the empty value field, the resulting include file would look like (shortened):

*-------------------------------------------------------
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
*-------------------------------------------------------

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

Set
i 'canning plants' / seattle, san-diego /
j 'markets' / new-york, chicago, topeka /;
$call mdb2gms I=Sample.mdb" Q="SELECT city1, city2, '' FROM distances" O=city_ij.inc

Set ij(i,j) 'two dimensional set' /
$include city_ij.inc
/;
display ij;

The display statement generates the following output in the listing file:

----  75 SET ij two dimensional set

   new-york  chicago  topeka
SAN-DIEGO  YES      YES      YES
SEATTLE    YES      YES      YES

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the X and S resp. Y arguments, i.e. there is no need to modify the query:

SELECT index1, index2, index3 FROM datatable
when using MDB2GMS in the following way:

mdb2gms I=Sample.mdb Q="SELECT index1, index2, index3 FROM datatable" X=setData.gdx S=setName

which will generate the file setData.gdx with a three dimensional set named setName containing the results of the query.
6.32.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a MS Access database file using MDB2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the Y argument in order to store the query results as a set with explanatory text in a GDX file.

![setData](image)

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```
SELECT setElement, explText
FROM setData
```

**Writing the Query Results in an include File**

The last column in the select clause will be used as explanatory text. Take in mind to add the argument B in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```
$call mdb2gms I=Sample.mdb B Q="SELECT setElement, explText FROM setData" O=setData.inc
Set a /
$include setData.inc
/;
```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
firstSetElement "Explanatory text for the first 'set element'
secondSetElement "Explanatory text for the second "set element"
thirdSetElement "Explanatory text for the third 'set element'
fourthSetElement "Explanatory text for the fourth set element"
* -----------------------------------------------------
```

Note the handling of the quotes according to the description in B.

**Writing the Query Results in a GDX File**

When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the Y argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file setData.inc:

```
$call mdb2gms I=Sample.mdb Q="SELECT setElement, explText FROM setData" X=setData.gdx Y=set_b
Set b;
.gdxIn setData.gdx
$load b = set_b
.gdxIn
```
Note that the name of the set in the GDX file is `set_b` (specified within the Y argument), while the name of the GDX file was specified within the X argument.

### 6.32.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```gams
Set i / NY, DC, LA, SF /;
```

Now assume a data table looks like:

![Example Table](image)

This means we have to map ‘new york' to ‘NY' etc. This mapping can be done in two places: either in GAMS or in the database.

#### 6.32.3.7.1 Index Mapping done in GAMS

When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

```plaintext
* -----------------------------------------------------
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----------------------------------------------------
```

Note that the single quotes are added by activating the option `B` (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:
Set i / NY, DC, LA, SF /;
Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;
$call mdb2gms I=Sample.mdb B O="city1.inc" Q="SELECT city, value FROM [example table]"
Parameter dbdata(idb) /
$include city1.inc /
Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;
Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;

The display statement generates the following output in the listing file:

----- 47 PARAMETER data
NY 100.000, DC 102.000, LA 120.000, SF 105.000

This example is also part of the GAMS Data Utilities Library, see model [IndexMapping1] for reference.

6.32.3.7.2 Index mapping done in Database  The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

![CityMapper](image)

**Figure 6.52 Table: CityMapper**

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT [GAMS City], value
FROM example_table, CityMapper
WHERE CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
la 120
ny 100
sf 105
dc 102
* -----------------------------------------------------
```
All steps can be combined in the GAMS model:

Set i / NY, DC, LA, SF /;

$onEcho > howToRead.txt
I=Sample.mdb
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_table.city
O=city2.inc
$offEcho

$call mdb2gms @howToRead.txt
$include city2.inc
Parameter data(i) /

The display statement generates the following output in the listing file:

----- 38 PARAMETER data
NY 100.000, DC 102.000, LA 120.000, SF 105.000

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model [IndexMapping2] for reference.

6.32.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual MDB2GMS execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single MDB2GMS call and write several GAMS include files containing the results of the queries, we can use the command-line arguments Qn and On. The structure of a multi-query call looks like:

I=sample.mdb

Q1=firstQuery
O1=outputFileName.inc

Q2=secondQuery
O2=outputFileName.inc

Q3=thirdQuery
O3=outputFileName.inc

The terms firstQuery, secondQuery etc. are placeholders for some SQL statements. We see that the argument Qn is matched by an argument On. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments Qn, Sn, Pn and Yn. The structure of a multi-query call when writing to a GDX file looks like:
I=sample.mdb
X=sample.gdx

Q1=firstQuery
S1=setName

Q2=secondQuery
S2=setName

Q3=thirdQuery
A3=parameterName

Q4=fourthQuery
A4=setName

Again, the terms firstQuery, secondQuery etc. are placeholders for some SQL statements. Here we see that a query Qn is matched by either a set name Sn or a parameter name Pn, i.e. the results of the first query will be stored as a set whose name is specified within the S1 argument, the results of the third query will be stored as a parameter whose name is specified within the P3 argument etc. The X argument is used to specify the name of the GDX file.

For a complete example see section Example 7 - Multi-Query Batch Example.

6.32.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file Sample.mdb):

![Table: data](image)

Figure 6.53 Table: data (shortened for presentation)

We want to extract the following information:

- The set year
- The set loc
• The set \texttt{prd}
• The parameter \texttt{sales}
• The parameter \texttt{profit}

**Output: Several include Files**

This can be accomplished using the following GAMS code with multiple queries in a single \texttt{MDB2GMS} call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):

```gams
$onEcho > howToRead.txt
I=Sample.mdb
Q1=SELECT distinct(year) FROM data
O1=year.inc
Q2=SELECT distinct(loc) FROM data
O2=loc.inc
Q3=SELECT distinct(prod) FROM data
O3=prod.inc
Q4=SELECT prod, loc, year, sales FROM data
O4=sales.inc
Q5=SELECT prod, loc, year, profit FROM data
O5=profit.inc
$offEcho
$call =mdb2gms @howToRead.txt
Set y 'years' /
$include year.inc
/;
Set loc 'locations' /
$include loc.inc
/;
Set prd 'products' /
$include prod.inc
/;
Parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;
Parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

This example is also part of the GAMS Data Utilities Library, see model \texttt{[SalesProfitDB3]} for reference.

**Output: A single GDX File**

The same example imported through a GDX file can look like:
The call of the GDXViewer will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model [SalesProfitDB4] for reference.

### 6.32.5 Interactive Usage

When the tool is called without command-line parameters, it will startup interactively. Using it this way, one can specify the database file (.mdb or .accbd file), the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.
• **Input file (.mdb or .accbd)**. This is the combo box to specify the input file. See also `inputFile` for some more detailed remarks. The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\machine\directory\file.mdb`.

  ![Input File](image)

  **Figure 6.54 MDB2GMS - Graphical User Interface**

  ![Image](image)

• **Output GAMS Include file (.inc)**. If you want to create a GAMS include file, then specify here the destination file. See also `outputIncludeFile` for some more detailed remarks.

  ![Output File](image)
• **Output GDX file** (*.gdx*). As an alternative to a GAMS include file, the tool can also generate a **GDX file**. One or both of the output files need to be specified. See also **outputGDXFile** for some more detailed notes.

```
Output: GDX file (*.gdx)
                      Browse...
                      E:\gam\model\distances.gdx
```

• **SQL Query**. The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also **Q** for some more detailed remarks. For an exact description of allowed expressions consult a text on MS Access.

```
SQL query
SELECT city1, city2, distance
FROM   distances
```

• **Progress Memo**. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.

```
DB2 version: 11.0
Jet version: 4.0
Database: Y:\stj\ust\My Documents\gamsdir\projdir\Sample.mdb
Options: R=100 V=0
Query: SELECT city1, city2, distance
       FROM   distances
Number of rows: 6
Elapsed time: 0.03 seconds
```

• The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

• The **Tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.
• The **Options button** will pop up a window where you can specify a number of options.

• Pressing the **Help button** will show this documentation.

• Pressing the **OK button** will execute the query and an include file or GDX file will be generated.

• Pressing the **Batch button** will give information on how the current query can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied to the clipboard. In the IDE press **Ctrl-C** or choose **Edit|Paste** to copy the contents of the clipboard to a GAMS text file.

• Pressing **Close button** will exit the application. The current settings will be saved in an INI file so when you run **MDB2GMS** again all current settings will be restored.

6.32.5.1 **Options**

The **Options** window can be created by pressing the options button:
The following options are available in the options window:

- **Quote blanks**: Quote strings if they contain blanks or embedded quotes. See also B for some more detailed notes.
  
  ![Quote blanks checkbox]

- **Mute**: Don’t include the extra informational text (such as used query etc.) in the include file.
  
  ![Mute checkbox]

- **No listing**: Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is L.
  
  ![No listing checkbox]

- **Format SQL**: If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.
The following options are only needed in special cases:

- **NULL**: This radio box determines how NULL’s are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL’s the default to throw an exception is a safe choice. In special cases you may want to map NULL’s to an empty string or a 'NULL' string.

  ![NULL Options](image)

- **Output Lines**: By default output lines are created as follows: all first n-1 fields are considered indices and the last n-th column is the value. The format corresponding to this situation is ‘%s,%s,%s %s’ (for a three dimensional parameter). In special cases you may want to tinker with the format string being used. The fields are all considered strings, so only use s as format placeholder. Make sure you specify exactly the same number of s’s as there are columns in the result set.

  ![Output Lines Options](image)

The buttons have an obvious functionality:

- **OK button** will accept the changes made.

  ![OK Button](image)

- **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.
• Help button will show this help text.

6.32.6 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

• The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.

• There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.

• It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing

we can conveniently use the save/restart facility. The individual submodel are coded as:

• Step 0: sr0.gms

```
$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
   i 'suppliers'
   j 'demand centers';

Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';
```
$onEcho > howtoRead.txt
I=transportation.mdb

Q1=select name from suppliers
O1=i.inc

Q2=select name from demandcenters
O2=j.inc

Q3=select name,demand from demandcenters
O3=demand.inc

Q4=select name,supply from suppliers
O4=supply.inc

Q5=select supplier,demandcenter,distance from distances
O5=dist.inc
$offEcho

$call =mdb2gms.exe @howtoRead.txt

Set i /
$include i.inc
/;

Set j /
$include j.inc
/;

Parameter demand /
$include demand.inc
/;

Parameter supply /
$include supply.inc
/;

Parameter dist /
$include dist.inc
/;

display i, j, demand, supply, dist;

• Step 1: sr1.gms

$onText
gms step 1: data manipulation
execute as: > gams sr1 restart=s0 save=s1
$offText

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;

• Step 2: sr2.gms

$onText
gms step 2: model definition
Variable
  x(i,j) 'shipment quantities in cases'
  z    'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
  ecost     'define objective function'
  esupply(i) 'observe supply limit at plant i'
  edemand(j) 'satisfy demand at market j';

  ecost..   z =e= sum((i,j), c(i,j)*x(i,j));
  esupply(i).. sum(j, x(i,j)) =l= supply(i);
  edemand(j).. sum(i, x(i,j)) =g= demand(j);

• Step 3: sr3.gms

  $onText
  step 3: model solution
  execute as: > gams sr3 restart=s2 save=s3
  $offText

  option lp = cplex;

  Model transport / all /;
  solve transport using lp minimizing z;

• Step 4: sr4.gms

  $onText
  step 4: report writing
  execute as: > gams sr4 restart=s3
  $offText

  abort$(transport.modelStat <> 1) "model not solved to optimality";

  display x.l, z.l;

A model that executes all steps can be written as:

execute '=gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";
If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [MDBSr5] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

**MDB2GDX1.gms**

```
execute '=gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorLevel "step 0 failed";
execute '=gdxviewer.exe trnsport.gdx';
```

The model [MDB2GDX1] is also part of the GAMS Data Utilities Library.

We can also let MDB2GMS create the GDX file:

**MDB2GDX2.gms**

This model demonstrates how to store data from Access database (file Transportation.mdb) into a GDX file.

```
$onEcho > howToRead.txt
I=Transportation.mdb
X=Transportation.gdx
Q1=SELECT name FROM suppliers
   S1=i
Q2=SELECT name FROM demandcenters
   S2=j
Q3=SELECT name, demand FROM demandcenters
   P3=demand
Q4=SELECT name, supply FROM suppliers
   P4=supply
Q5=SELECT supplier, demandcenter, distance FROM distances
   P5=dist
$offEcho
$call =mdb2gms.exe @howToRead.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

The model [MDB2GDX2] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

**GDXTRNSPORT.gms**

This model demonstrates how to load the transportation data from GDX file at compile time.
Set
  i 'suppliers'
  j 'demand centers';

Parameter
  demand(j)
  supply(i)
  dist(i,j) 'distances';

$gdxIn transportation.gdx
$load i j demand supply dist
display i, j, demand, supply, dist;

DBTimestamp1.gms

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

$onText
Retrieve data from database first run each morning.
$offText

$onEcho > getdate.txt
I=%system.fp%transportation.mdb
Q=select day(now())
O=dbtimestamp.inc
$offEcho

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc /
Scalar currentday 'day of this run';
currentday = gday(jnow);
display "compare", dbtimestamp, currentday;
if(dbtimestamp <> currentday,
  execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
  abort$errorLevel "step 0 (database access) failed";
  execute '=mdb2gms.exe @getdate.txt' );

The include file dbtimestamp.inc contains the day of the month (1,...,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

The model [DBTimestamp1] is also part of the GAMS Data Utilities Library.
6.32.7 Command Files

Parameters can be specified in a command file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command-line as in:

```gams
$call =mdb2gms I="c:\My Documents\test.mdb" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command-line like:

```gams
$call =mdb2gms @"c:\My Documents\options.txt"
```

The command file

```
c:\My Documents\options.txt
```

can look like:

```
I=c:\My Documents\test.mdb
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable
```

It is possible to write the command file from inside a GAMS model using the `$echo` command. The following example will illustrate this:

```gams
$set cmdfile "c:\windows\temp\commands.txt"
$echo "I=E:\models\labordata.mdb" > "%cmdfile%"
$echo "O=E:\models\labor.INC" >> "%cmdfile%"
$echo "Q=SELECT * FROM labor" >> "%cmdfile%"
$call =mdb2gms @"%cmdfile%"
```

Parameter p /

```gams
$include "E:\models\labor.INC"
```

```
/;
```

Display p;

Newer versions of GAMS allow the usage of the `$onEcho` and `$offEcho` commands:

```gams
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho > "%cmdfile%"
I=E:\models\labordata.mdb
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho
$call =mdb2gms @"%cmdfile%"
```

Parameter p /

```gams
$include "E:\models\labor.INC"
```

```
/;
```

Display p;
Note that the quotes enclosing strings with blanks like Q=SELECT ∗ FROM labor can be omitted when using an external parameter file.

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

Q=SELECT prod, loc, year, 'sales', sales FROM data \  
UNION \  
SELECT prod, loc, year, 'profit', profit FROM data

6.32.8 Notes

6.32.8.1 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: $gdxIn, $load, execute_load, execute_unload. The GDX=filename command-line argument will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using File|Open.

6.32.8.2 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

\<server>\<share>\<path>\<file>

Examples:

\athlon\c\My Documents\MDB2GMS.rtf

6.32.8.3 Quotes

Examples of handling of indices when the option B for quoting strings containing blanks is used:
6.32.8.4  $CALL Command

The $call command in GAMS will execute an external program at compile time. There are two forms:

$call externalProgram

$call =externalProgram

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form $call externalProgram which uses the command processor.

2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form $call externalProgram.

3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form $call =externalProgram. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: $call =externalProgram.

4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: $call start /w externalWindowsProgram. Otherwise, it is preferred to use: $call =externalWindowsProgram.

Attention

In general it is recommended to use the $call =externalProgram version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

$call "external program" "parameter 1" "parameter 2"
$call ="external program" "parameter 1" "parameter 2"

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes only if the '=' appears outside the double quotes.
6.32.8.5 Compile Time Commands

All $ commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed before an execution time command, even if it is below. As an example consider:

File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat

This fragment does not work correctly as already during compilation, the $call is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

$echo "dir" > x.bat
$call x.bat

or by moving the external program invocation to execution time:

File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;

Notice that all $ commands do not include a semi-colon but are terminated by the end-of-line.

6.33 MessageReceiverWindow

6.33.1 Introduction

MessageReceiverWindow is a graphical tool that receives and displays Windows messages. The tool can either be used with the put_utility feature or with an arbitrary client (e.g. a Python program) that sends messages to a previously spawned instance of MessageReceiverWindow.exe.

6.33.2 Usage

An instance of MessageReceiverWindow.exe can be started as follows:

MessageReceiverWindow <title>
The optional argument `title` specifies the title of the `MessageReceiverWindow` instance. If the argument is omitted, the instance gets the default title `Form1`. Windows messages of type `WM_COPYDATA` that are sent to the spawned instance, appear in its message log. The `Copy To Clipboard` button allows to copy the content of the message log to the clipboard. The `Save As ...` button can be used to save the content to a file.

![MessageReceiverWindow](image)

**Figure 6.56** MessageReceiverWindow during the execution of Test Library model `MRW01`.

### 6.33.3 Special Commands

While arbitrary text messages are displayed in the message log, there are special commands for controlling special behavior of an instance of `MessageReceiverWindow`:

<table>
<thead>
<tr>
<th>Message</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@CopyToClipboard</code></td>
<td>Copies the content of the message log to the clipboard</td>
</tr>
<tr>
<td><code>@SaveAs [file]</code></td>
<td>Writes the content of the message log to <code>[file]</code></td>
</tr>
<tr>
<td><code>@Terminate</code></td>
<td>Terminates the MessageReceiverWindow executable</td>
</tr>
</tbody>
</table>

### 6.33.4 Usage With `put_utility`

The `put_utility` feature offers a convenient integration of `MessageReceiverWindow`. The example `MRW01` shows how this works. First of all, an instance of `MessageReceiverWindow` needs to be spawned from within GAMS:

```gams
$set title Form1
Execute.Async "MessageReceiverWindow.exe %title%";
```
As soon as the window is ready, text messages or special commands can be sent by using put_utility with the command \texttt{winMsg}:

```gams
file f;
put_utility f 'WinMsg' / '%title%' / 'This is some Windows Message from GAMS';
```

By sending \texttt{@Terminate}, the instance can be terminated:

```gams
put_utility f 'WinMsg' / '%title%' / '@Terminate';
```

### 6.33.5 Usage With Python

The example \texttt{MRW02} from the GAMS Test Library demonstrates the use of \texttt{MessageReceiverWindow} from within Python. While it uses the Python programming language from within an Embedded Code section, other programming languages can be used as long as they provide access to Windows messages.

First of all, a ctypes.

```python
class CopyDataStruct(ctypes.Structure):
    _fields_ = [('dwData', ctypes.c_char),
                ('cbData', ctypes.c_ulong),
                ('lpData', ctypes.c_charp)]
```

The function \texttt{sendMessage()} sends a message to the window with the title \texttt{windowName}. It makes use of functions \texttt{FindWindowA} and \texttt{SendMessageA} from ctypes.windll.user32:

```python
def sendMessage(windowName, message):
    cs = CopyDataStruct()
    receiver = ctypes.windll.user32.FindWindowA(None, bytes(windowName, 'utf-8'))
    cmd = bytes(message, 'utf-8')
    cs.dwData = 1
    cs.cbData = len(cmd)+1
    cs.lpData = cmd
    WM_COPYDATA = 0x4A
    ctypes.windll.user32.SendMessageA(receiver, WM_COPYDATA, 0, ctypes.byref(cs))
```

\texttt{Popen} from the \texttt{subprocess} module is used in order to start an instance of \texttt{MessageReceiverWindow}. The \texttt{system\_directory} property of a GamsWorkspace instance from the GAMS Python API is used for finding a GAMS system directory:

```python
windowName = 'MRW Python'
p = Popen([os.path.join(GamsWorkspace().system\_directory, 'MessageReceiverWindow.exe'), windowName])
```

As soon as the window is ready, text messages or special commands can be sent by using the previously defined \texttt{sendMessage} function:

```python
sendMessage(windowName, "This is some Windows Message from Python")
```

Sending \texttt{@Terminate} will terminate the running \texttt{MessageReceiverWindow} instance:

```python
sendMessage(windowName, "@Terminate")
```

### 6.34 MODEL2TEX

A Tool to generate a documentation from GAMS source code in LaTeX format.

**Author**

Clemens Westphal

Special thanks to Ingmar Schlecht for supporting us and sharing his gamsToLatex implementation.
6.34.1 Introduction

MODEL2TEX is a tool to generate a documentation from GAMS source code in LaTeX format. This LaTeX output can then be further processed in order to generate pretty output files like PDF. The tool can be found in the root directory of GAMS. The tool allows to document one specific model symbol inside of a GAMS program. The resulting documentation contains two parts. The first part shows a list of symbols that are used by the model. The second part shows the actual algebra of the used equations and information about the types of used variables. An optional third part can contain additional notes.

6.34.2 Usage

MODEL2TEX is a command line tool. The general command line usage is as follows:

```
model2tex baseName [-h] [-m MODEL] [-f] [-o OUTPUT]
```

Since it operates on output files generated by GAMS, the first step is to generate the required files using the docfile option. In order to do that, execute the following command line:

```
gams myModel.gms docfile=myModel
```

The second step is to call MODEL2TEX: model2tex is distributed as Python source code (model2tex.py) and is executed via model2tex.cmd on Windows and model2tex.sh on all other platforms. For platforms other than Windows, Linux, or macOS, it is required to provide a Python 3 installation with access to the GAMS Python API.

Windows:

```
model2tex myModel [-m MODEL] [-f] [-o OUTPUT]
```

Linux/Mac OS X:

```
model2tex.sh myModel [-m MODEL] [-f] [-o OUTPUT]
```

The output file myModel.tex can be further processed for example by calling pdflatex in order to generate a PDF file.

```
pdflatex myModel.tex
```

6.34.3 Options

The following parameters can be used when calling MODEL2TEX:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-m model</code></td>
<td>Since MODEL2TEX generates documentation for one model symbol, the model name needs to be specified explicitly, if the GAMS file contains more than one model symbol.</td>
</tr>
<tr>
<td><code>-f</code></td>
<td>MODEL2TEX does not allow the occurrence of suffixes in equations. This option can be used in order to force the creation of the tex file by skipping the checks for suffixes.</td>
</tr>
<tr>
<td><code>-o output</code></td>
<td>Use this name for the generated TeX file instead of the base name.</td>
</tr>
</tbody>
</table>
6.34.4 Using a JSON style file

MODEL2TEX automatically creates a JSON file that can be modified in order to customize the output. If you want to get the default settings back, just delete the generated JSON file and let MODEL2TEX create it again. The following list shows the available options in the JSON file:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fontSize</td>
<td>11</td>
<td>The size of the used font</td>
</tr>
<tr>
<td>hrules</td>
<td>true</td>
<td>Horizontal rules are added between equations.</td>
</tr>
<tr>
<td>colors</td>
<td>black</td>
<td>Specifies the used colors in equations for variables, parameters, and sets.</td>
</tr>
<tr>
<td>landscape</td>
<td>false</td>
<td>Allows to change the page format to landscape.</td>
</tr>
<tr>
<td>noPowerFunc</td>
<td>false</td>
<td>Beside a power operator, GAMS offers several power functions. Setting this option to true will replace all power functions with the power operator.</td>
</tr>
<tr>
<td>reduceFrac</td>
<td>false</td>
<td>Try to resolve unbalanced fractions. This means that fractions with an unbalanced length of denominator and numerator will be changed in order to shrink the fraction.</td>
</tr>
<tr>
<td>reduceFracRatio</td>
<td>5.0</td>
<td>This number has only effect when reduceFrac is set to true. It specifies the ratio between denominator and numerator that is used as a threshold for reduceFrac.</td>
</tr>
<tr>
<td>latexDescription</td>
<td>false</td>
<td>Explanatory text for symbols is expected to be arbitrary text. In order to display the text correctly in LaTeX, some automatic adjustments are applied. If all explanatory text already contains valid LaTeX strings, this option can be enabled.</td>
</tr>
<tr>
<td>nameMap</td>
<td>the original names</td>
<td>This map allows to specify alternative names for symbols.</td>
</tr>
<tr>
<td>extraSymbols</td>
<td>empty list</td>
<td>On default, only the required symbols in a GAMS model symbol are contained in the generated LaTeX file. This list can be used in order to specify further symbols that should be included as well.</td>
</tr>
<tr>
<td>notes</td>
<td>empty list</td>
<td>Allows to specify additional notes for the documentation. Each list element will result in a new line.</td>
</tr>
</tbody>
</table>

6.34.5 Example

This example creates a PDF file for the pump model from the GAMS Model Library.

1. Retrieve model pump from GAMS Model Library
   
   gamslib pump

2. Generate required doc files
   
   gams pump.gms docfile=pump

3. Generate the LaTeX files
   
   model2tex pump -m=pump

4. Generate PDF documentation
   
   pdflatex pump.tex
The following pictures show snippets from the resulting PDF file:

5. Customizing the JSON style file In order to change the appearance of the PDF file, the generated JSON file `pump.json` can be modified. Changing the options in `pump.json` according to the picture and executing the commands from step 3 and 4 again will result in a customized PDF output.

```
"fontSize": 10,
"bRules": false,
"colors": {
  "variables": "red",
  "parameters": "blue",
  "sets": "black"
},
"landscape": false,
"noPowerFunc": true,
"reduceFrac": false,
"reduceFracRatio": 5.0,
"latexDescription": false,
"nameMap": {
  "vdot": "$\dot{v}$",
  "sumx": "sumx",
  ...
},
"extraSymbols": [],
"notes": []
```
Translates an MPS or LP file into an equivalent short generic GAMS program using a GDX file to store data. A number of solver specific extensions are recognized.

The MPS file is first attempted to be read in free format (no spaces in names). If it seems that row or column names have spaces but have at most 8 characters, parsing in fixed format is attempted. The MPS and LP readers of the HiGHS solver are utilized. GAMS Data Exchange (GDX) files and matching GAMS source is written.

Notes:

- Row, column, SOS (special ordered sets), and cone names can have up to 63 characters and are tested for case sensitivity and uniqueness because these names become GAMS set members. If the test fails, the tool attempts to modify the names.
- MPS files may contain multiple N rows. The first one will be used as objective function and remaining ones will be ignored.
- If no optimization direction is given in a MPS file, minimization will be assumed.
- Variables in SOS must be continuous.
- SOS must not overlap.

The MPS and LP formats have been extended in different forms by several solver vendors. Not all extensions are supported by MPS2GMS. The following MPS sections are recognized by the free format parser:
• NAME
• OBJSENSE, MAX, MIN
• ROWS
• COLUMNS
• RHS
• BOUNDS
• RANGES
• QUADOBJ, QMATRIX
• QSECTION, QCMATRIX
• CSECTION with cone types QUAD and RQUAD
• SETS, SOS
• ENDDATA

The fixed format parser also recognizes these sections except for QMATRIX, QSECTION, QCMATRIX, CSECTION, SETS, and SOS.

For a description of the MPS format, see also

• the CPLEX manual,
• the FICO Xpress manual,
• the GUROBI manual,
• the MOSEK manual.

The following LP format features are recognized:

• Comments
• Single objective function: linear or quadratic only
• Constraints (Subject To section): linear or quadratic only
• Variable bounds and types (Bounds, Binaries, Generals, Semis section): to specify semi-integer variables, they need to appear in sections Generals and Semis
• Special ordered sets (SOS section)

For a description of the LP format, see also

• the CPLEX manual
• the FICO Xpress manual
• the GUROBI manual
• the MOSEK manual

6.35.1 Usage

mps2gms mpsfile | lpfile [ gdxfile [ gmsfile ] ] { key=value }

If the first parameter is not of the form key=value and ends with .lp or .lp.gz, it is assumed to be the name of an LP file, otherwise it is assumed to be the name of an MPS file. If no GDX filename is given, the name of the MPS or LP file is used, with .mps/.lp (or .mps.gz/.lp.gz) extension replaced by .gdx, or .gdx appended. If no GMS filename is given, the name of the GDX file is used, with .gdx extension replaced by .gms, or .gms appended.

Guide to parameters:
### Key Description

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPS</td>
<td>Name of MPS input file, can be compressed with GZIP</td>
</tr>
<tr>
<td>LP</td>
<td>Name of LP input file, can be compressed with GZIP</td>
</tr>
<tr>
<td>GDX</td>
<td>Name of GDX output file</td>
</tr>
<tr>
<td>GMS</td>
<td>Name of GAMS program output file</td>
</tr>
<tr>
<td>CEQUATIONS</td>
<td>Whether to write quadratic cones from CSECTION sections into deprecated =C= equations. Possible values: 0, N, 1, Y. Default: 0</td>
</tr>
<tr>
<td>COLUMNINTVARSAREBINARY</td>
<td>Whether integer variables that appear first in COLUMNS section should be assumed to be binary variables. Possible values: 0, N, 1, Y. Default: 0. Note that modern solvers assume binary variables, but for backward compatibility the default is set to the original IBM interpretation.</td>
</tr>
</tbody>
</table>

### 6.36 MSAPPAVAIL

Checks if a MS Office Application is available.

#### 6.36.1 Usage

`msappavail [option]`

**Option:**

```
?
```

List all known applications and their status. Check for specific application (ExitCode = 0 when available)

- Access
- Excel
- Explorer
- FrontPage
- Outlook
- PowerPoint
- Project
- Word

See how the example from the GAMS model library uses this feature.
6.37 GAMS Posix Utilities

Starting with distribution 20.6 the GAMS system for Windows includes a collection of Posix utilities which are usually available for the different Unix systems and therefore help to write platform independent scripts. The following utilities are available:

- awk - Pattern scanning and processing language
- cat - Concatenate and print files
- cksum - Write file checksums and sizes
- cmp - Compare two files
- comm - Select or reject lines common to two files
- cp - Copy files
- cut - Cut out selected fields of each line of a file
- diff - Compare two files
- expr - Evaluate arguments as an expression
- fold - Fold lines
- gsort - Sort, merge, or sequence check text files
- grep - File pattern searcher
- gdate - Write the date and time
- head - Copy the first part of files
- join - Relational database operator
- mv - Move files
- od - Dump files in various formats
- paste - Merge corresponding or subsequent lines of files
- printf - Write formatted output
- rm - Remove directory entries
- sed - Stream editor
- sleep - Suspend execution for an interval
- tail - Copy the last part of a file
- tee - Duplicate standard input
- test - Evaluate expression
- tr - Translate characters
- uniq - Report or filter out repeated lines in a file
- wc - Word, line, and byte count
- xargs - Construct argument list(s) and invoke utility

Please note that the utilities "date" and "sort" have been renamed to "gdate" and "gsort" to avoid conflicts with the Windows commands "date" and "sort". For compatibility reasons the GNU implementation of awk called "gawk" has been renamed to "awk".

The collection consists of native Windows ports of GNU implementation of these utilities taken from the Sourceforge. Detailed descriptions of the utilities can be found at the GNU website.

The Posix tools and in particular the awk utility can be used to transform a variety of different text inputs into GAMS readable input files. Examples can be found in the GAMS Model Library (Subject "GAMS Tools").
6.38 SCENRED

6.38.1 Release Notes

• May, 2002: Level 001 (GAMS Distribution 20.6)
  – GAMS/SCENRED introduced.

6.38.2 Introduction

Stochastic programs with recourse employing a discrete distribution of the random parameters become a deterministic programming problem. They can be solved by an appropriate optimization algorithm, ignoring the stochastic nature of (some or all) parameters.

SCENRED is a tool for the reduction of scenarios modeling the random data processes. The scenario reduction algorithms provided by SCENRED determine a scenario subset (of prescribed cardinality or accuracy) and assign optimal probabilities to the preserved scenarios. The reduced problem is then solved by a deterministic optimization algorithm provided by GAMS.

6.38.3 Scenario Reduction Algorithms

Many solution methods for stochastic programs employ discrete approximations of the uncertain data processes by a set of scenarios (i.e., possible outcomes of the uncertain parameters) with corresponding probabilities.

For most practical problems the optimization problem that contains all possible scenarios (the so-called deterministic equivalent program) is too large. Due to computational complexity and to time limitations this program is often approximated by a model involving a (much) smaller number of scenarios.

The reduction algorithms developed in [1,2] determine a subset of the initial scenario set and assign new probabilities to the preserved scenarios. All deleted scenarios have probability zero.

SCENRED contains three reduction algorithms: The Fast Backward method, a mix of Fast Backward/Forward methods and a mix of Fast Backward/Backward methods. In general, the computational performance (accuracy, running time) of the methods differ. For huge scenario trees the Fast Backward method has the best expected performance with respect to running time. The results of the Forward and Backward methods are more accurate, but at the expense of higher computing time. The Forward method is the best algorithm when comparing accuracy, but it can only be recommended if the number of preserved scenarios is small (strong reduction). The combined methods improve the result of the Fast Backward method if the Forward or Backward method, respectively, can be completed within the running time limit. If no reduction method is selected, the method with the best expected performance with respect to running time is chosen.

The reduction algorithms exploit a certain probability distance of the original and the reduced probability measure. The probability distance trades off scenario probabilities and distances of scenario values. Therefore, deletion will occur if scenarios are close or have small probabilities.

The reduction concept is general and universal. No requirements on the stochastic data processes (e.g. the dependency or correlation structure of the scenarios, the scenario probabilities or the dimension of the process) or on the structure of the scenarios (e.g. tree-structured or not) are imposed. The reduction algorithms can be tailored to the stochastic model if the user provides additional information (How many decision stages are involved? Where do the random parameters enter the model – in objective and/or right hand sides and/or technology matrices?) The information is used to choose the probability distances (cf. Remark 1 in [1]).

References:


6.38.4 Using GAMS/SCENRED

The reduction algorithms require additional data preparation and reformulation of the GAMS program for the stochastic programming model.

GAMS offers great flexibility with respect to the organization of data specification, model definition and solve statements. The most common way to organize GAMS/SCENRED programs is shown below. Since the initial scenarios and a number of input parameters have to be passed to SCENRED, the corresponding components of the GAMS program have to be defined before the SCENRED call. The reduced scenarios have to be defined before the equations of the (reduced) stochastic programming model are used in a solve statement. Therefore the SCENRED call can be placed anywhere between the definitions of the GAMS parameters and the solve statement of the reduced stochastic programming model.

When building or modifying a model for use with GAMS/SCENRED the following steps should be taken:

- Analyse the GAMS program of the stochastic programming model.
  Since the initial scenarios and a number of input parameters have to be passed to SCENRED (see Section The SCENRED Input File), one must identify the corresponding components of the GAMS model and create or calculate them if they do not already exist.

- Reformulate the GAMS program.
  Check if the model can handle varying scenario or node probabilities, and whether the equations are defined in terms of a (possibly reduced) tree. If the model doesn't already contain a scenario tree, one should be added. If it does, it is a simple task to rewrite the equation definitions (and possibly other statements too) in terms of a subset of the original nodes or tree.

- Add the statements for passing the initial set of scenarios to SCENRED, for the execution of SCENRED and for the import of the reduced scenarios from SCENRED.

A reduction of the initial scenarios makes sense only if we are able to generate that part of the model that corresponds to the preserved scenarios (i.e. the reduced subtree). This is done by declaring a subset of the nodes in the original tree. The parameters and equations are declared over the original node set, but are defined over only the subtree. This will be illustrated by an example later in the section.

Further, one should verify that the model can handle changing probabilities. Many practical models involve scenarios with equal probabilities. This property will not be maintained by the probabilities in the reduced subtree.

**ORGANIZATION OF GAMS/SCENRED PROGRAMS**

<table>
<thead>
<tr>
<th>Component</th>
<th>Contents</th>
</tr>
</thead>
</table>
| 1. DATA   | * set & parameter declarations and definitions  
|           | * $libinclude scenred.gms  
|           | * assignments displays |
| 2. SCENRED CALL | * export the initial scenarios from GAMS to SCENRED  
|                | * execute SCENRED  
|                | * import the reduced scenarios from SCENRED to GAMS |
| 3. MODEL   | * variable declaration  
|           | * equation declarations  
|           | * equation definitions (using sets from reduced tree)  
|           | * model definition & solution |

Prior to calling SCENRED, you should include the declaration of the SCENRED input and output parameters and the definition of the sets they are indexed by from the GAMS include library:
Once you have created all the inputs to SCENRED and assigned values to them, you are ready to write the SCENRED GDX data input file, write the SCENRED options file, call SCENRED, and read the reduced tree data from the SCENRED GDX data output file (see Section The SCENRED Input File, Section SCENRED Options and the Option File, and Section The SCENRED Output File). Assuming your model is formulated to use a subtree of the original, you can now continue with the solve and any subsequent reporting.

SCENRED is executed by issuing the statement

```
execute 'scenred optfilename';
```

where `optfilename` is the name of the SCENRED option file.

As an example, consider the `srkandw` model in the GAMS model library, and the `kand` model upon which it is based (get these from the modlib now!). To produce `srkandw` from `kand`, we first reformulate the original to allow for solution over a reduced tree. To do this, we introduce a subset of the node set: `set sn(n) 'nodes in reduced tree'`; For convenience and clarity, we introduce a second subset at the same time, the set of leaf nodes: `set leaf(n) 'leaf nodes in original tree'`; as well as some code to compute this set based on the existing time-node mapping. We also declare a new parameter, the probabilities for the reduced tree: `parameter sprob(n) 'node probability in reduced tree'`; Once these are declared, we can quickly edit the equation definitions so that they run only over the reduced subtree: we simply substitute the reduced probabilities `sprob` for the original `prob`, and the reduced node set `sn` for the original node set `n`. Note that the declaration of the equations does not change.

This example illustrates one other change that may be required: the stochastic data must be in parameters having the node set as their last index. This is not the case in the `kand` model, so we simply reversed the indices in the `dem` parameter to meet the requirement in `srkandw`. It is also possible to create a transposed copy of the original data and pass that the SCENRED if the original data cannot be changed conveniently.

### 6.38.5 The SCENRED Input File

The SCENRED input file contains the initial scenarios and their stochastic parameter data, as well as statistics describing this input and (possibly) options to control the SCENRED run. This input file has a special binary format; it is a GDX (GAMS Data Exchange) file. The name of the SCENRED input file is assigned in the option file (see Section SCENRED Options and the Option File).

The scalar inputs to SCENRED are collected in the one-dimensional parameter `ScenRedParms`, the first parameter stored in the SCENRED input file. Some of the elements of `ScenRedParms` are required (e.g. statistics for the input tree) while others are optional (e.g. the run time limit). SCENRED will stop if a required element is missing or out of range.

#### Table 1 Required ScenRedParms elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_leaves</td>
<td>the number of initial scenarios or leaves of the scenario tree (i.e., before the reduction)</td>
</tr>
<tr>
<td>num_nodes</td>
<td>number of nodes in the initial tree (the number of scenarios if not tree-structured)</td>
</tr>
<tr>
<td>num_random</td>
<td>Number of random variables assigned to a scenario or node, i.e., the dimension of the random data process</td>
</tr>
<tr>
<td>num_time_steps</td>
<td>Length of a path from the root node to a leaf of the scenario tree, i.e., the number of time steps involved</td>
</tr>
</tbody>
</table>
### Table 2 Optional ScenRedParms elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>red_num_leaves</td>
<td>specifies the desired number of preserved scenarios or leaves</td>
<td>none</td>
</tr>
<tr>
<td>red_percentage</td>
<td>specifies the desired reduction in terms of the relative distance between the initial and reduced scenario trees (a real between 0.0 and 1.0)</td>
<td>none</td>
</tr>
<tr>
<td>num_stages</td>
<td>Set the number of branching levels of the scenario tree, i.e., the number of stages of the model -1. Hence num_stages=1 if no branching occurs, i.e., the values of the scenarios differ for all time steps</td>
<td>1</td>
</tr>
<tr>
<td>where_random</td>
<td>An integer indicating where the randomness enters the model. The value is interpreted as a digit map computed using the formula $100 \times \text{inObj} + 10 \times \text{inRHS} + \text{inMatrix}$, where inObj is 1 if the objective contains random parameters and 0 otherwise, inRHS is 1 if the right-hand side contains random parameters and 0 otherwise, and inMatrix is 1 if the constraint matrix contains random coefficients and 0 otherwise.</td>
<td>10 (random right-hand side)</td>
</tr>
<tr>
<td>reduction_method</td>
<td>Select a reduction method: 0: automatic (best expected performance with respect to running time) 1: Fast Backward method 2: Mix of Fast Backward/Forward methods 3: Mix of Fast Backward/Backward methods</td>
<td>0</td>
</tr>
<tr>
<td>run_time_limit</td>
<td>Defines a limit on the running time in seconds</td>
<td>none</td>
</tr>
<tr>
<td>report_level</td>
<td>Control the content of the SCENRED log file: 0: Standard SCENRED log file 1: Additional information about the tree</td>
<td>0</td>
</tr>
</tbody>
</table>

A few comments on the parameters `red_percentage` and `red_num_leaves` are in order. At least one of these values must be set. The value of `red_percentage` will be ignored if the parameter `red_num_leaves` is non-zero. Otherwise, the tree will not be reduced if `red_percentage`=0, while the reduction of the tree will be maximal (i.e. only one scenario will be kept) if `red_percentage`=1. A numeric value of 0.5 means that the reduced tree maintains 50% of the information contained in the original tree. The reduction algorithms are skipped if `red_num_leaves=num_leaves` or if `red_num_leaves=0` and `red_percentage=0`. These values can be assigned if the user wishes to run the scenario tree diagnostic.

The second data element in the input file is the set of nodes making up the scenario tree. Note that the cardinality of this set is part of ScenRedParms.

The third data element is the ancestor mapping between the nodes. This mapping determines the scenario tree. Note that the mapping can be either an ancestor mapping (i.e. child-parent) or a successor mapping (parent-child). By default, SCENRED expects an ancestor mapping. If the check for this fails, it looks for a successor mapping.

The fourth data element is the parameter of probabilities for the nodes in the original tree. It is only required that probabilities for the scenarios (i.e. the leaf nodes) be provided, but the parameter can contain probabilities for the non-leaf nodes as well.

The remaining elements in the input data file specify the parameter(s) that comprise the random values assigned to the initial scenarios, or to the nodes of the scenario tree. There can be more than one such...
parameter, included in any order. The only requirement is that the node set be the final index in each of these parameters.

Table 3 summarizes the content of the SCENRED input file. Please keep in mind that the order of the entries must not be altered!

Table 3 Content of the SCENRED Input File

<table>
<thead>
<tr>
<th>No.</th>
<th>Symbol</th>
<th>Type</th>
<th>Dimension</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ScenRedParms</td>
<td>Parameter</td>
<td>1</td>
<td>scalar SCENRED input</td>
</tr>
<tr>
<td>2</td>
<td>(any name)</td>
<td>Set</td>
<td>1</td>
<td>nodes in the scenario tree</td>
</tr>
<tr>
<td>3</td>
<td>(any name)</td>
<td>Set</td>
<td>2</td>
<td>the ancestor set</td>
</tr>
<tr>
<td>4</td>
<td>(any name)</td>
<td>Parameter</td>
<td>1</td>
<td>node probabilities; at least for the leaves</td>
</tr>
<tr>
<td>≥ 5</td>
<td>(any name)</td>
<td>Parameter</td>
<td>≥ 1</td>
<td>random values assigned to the nodes</td>
</tr>
</tbody>
</table>

To create the SCENRED data input file, the GAMS `execute_unload` statement is used. This statement is used to transfer GAMS data to a GDX file at execution time. As an example, to create a GDX file with the 4 required input parameters and one parameter `demand` containing the stochastic data, you might have the following statement:

```plaintext
eexecute_unload 'sr_input.gdx', ScenRedParms, node, ancestor, prob, demand;
```

6.38.6 SCENRED Options and the Option File

When the SCENRED executable is run, it takes only one argument on the command line: the name of the SCENRED option file. The option file is a plain text file. Typically, it is used to specify at least the names of the SCENRED data input and output files. The option file must be created by the SCENRED user (typically via the GAMS put facility during the GAMS run). The syntax for the SCENRED option file is

```plaintext
optname value or optname = value
```

with one option on each line. Comment lines start with an asterix and are ignored.

Some of the SCENRED options may be specified in two places: as elements of the `ScenRedParms` parameter of the SCENRED input file, or as entries in the options file. These parameters have been summarized in Table 2. If an option is set in both these places, the value in the option file takes precedence of over the value from `ScenRedParms`. In addition, the parameters in Table 4 can only be specified in the option file.

Table 4 Options - optfile only

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>input.gdx</td>
<td>Name of the SCENRED data input file</td>
<td>xlink.gdx</td>
</tr>
<tr>
<td>output.gdx</td>
<td>Name of the SCENRED data output file</td>
<td>scenred.gdx</td>
</tr>
<tr>
<td>log_file</td>
<td>Name of the SCENRED log file</td>
<td>scenred.log</td>
</tr>
</tbody>
</table>

6.38.7 The SCENRED Output File

The SCENRED output file contains the reduced scenario tree and the `ScenRedReport` parameter. Like the input file, the output file has a special binary format; it is a GDX (GAMS Data Exchange) file.
The first data element in the output file is the \texttt{ScenRedReport} parameter containing the scalar outputs and statistics from the SCENRED run. The elements of this parameter are summarized in Table 5. The second data element is the parameter containing the probabilities of the nodes in the reduced scenario tree. These node probabilities are required to construct the reduced tree. The third and final data element is the ancestor map for the reduced scenario tree. This map can be read from the GDX file, or the reduced tree can be built from the original one by using the reduced probabilities. The content of the data output file is summarized in Table 6.

\textbf{Table 5} \texttt{ScenRedReport} elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ScenRedWarnings}</td>
<td>number of SCENRED warnings</td>
</tr>
<tr>
<td>\texttt{ScenRedErrors}</td>
<td>number of SCENRED errors</td>
</tr>
<tr>
<td>\texttt{run_time}</td>
<td>running time of SCENRED in sec.</td>
</tr>
<tr>
<td>\texttt{orig_nodes}</td>
<td>number of nodes in the initial scenario tree</td>
</tr>
<tr>
<td>\texttt{orig_leaves}</td>
<td>number of leaves (scenarios) in the initial tree</td>
</tr>
<tr>
<td>\texttt{red_nodes}</td>
<td>number of nodes in the reduced scenario tree</td>
</tr>
<tr>
<td>\texttt{red_leaves}</td>
<td>number of leaves (scenarios) in the reduced tree</td>
</tr>
<tr>
<td>\texttt{red_percentage}</td>
<td>relative distance of initial and reduced tree</td>
</tr>
<tr>
<td>\texttt{red_absolute}</td>
<td>absolute distance between initial and reduced</td>
</tr>
<tr>
<td>\texttt{reduction_method}</td>
<td>reduction method used:</td>
</tr>
<tr>
<td></td>
<td>0: the program stopped before it could select a</td>
</tr>
<tr>
<td></td>
<td>method</td>
</tr>
<tr>
<td></td>
<td>1: Fast Backward method</td>
</tr>
<tr>
<td></td>
<td>2: Mix of Fast Backward/Forward methods</td>
</tr>
<tr>
<td></td>
<td>3: Mix of Fast Backward/Backward methods</td>
</tr>
</tbody>
</table>

To read the SCENRED data output file, the GAMS \texttt{execute_load} statement is used. This statement is used to transfer GDX data to GAMS at execution time. As an example, to read a GDX file named \texttt{sr_output.gdx} created by SCENRED, you might have the following statement:

\begin{verbatim}
execute_load 'sr_output.gdx', ScenRedReport, sprob=red_prob, sanc=red_ancestor;
\end{verbatim}

In the statement above, the equal sign = is used to indicate that the data in the GDX parameter \texttt{red_prob} should be read into the GAMS parameter \texttt{sprob}, and the data in the GDX set \texttt{red_ancestor} should be read into the GAMS set \texttt{sanc}.

\textbf{6.38.8 Diagnostic Check of Scenario Trees}

When SCENRED reads its input data, it performs a number of checks to verify that the data is correct. The diagnostic checks of the input parameters include:

- consistency of the desired input parameters with the contents of the SCENRED input file (number of nodes, number of leaves, number of time steps, number of random values assigned to a node)
• range check of desired input parameters and options
• check of scenario and node probabilities
• check of the ancestor matrix (check the orientation of the graph, check if the graph contains a cycle, check if the graph contains incomplete forests or scenarios, check the consistency of the parameter \texttt{num\_time\_steps} with the ancestor matrix)

The following errors in the specification of the scenario tree cause SCENRED to skip the reduction algorithms:

• The input files cannot be opened.
• Not all required input parameters are given.
• The required input parameters are not consistent with the contents of the SCENRED input file.
• The required input parameters are out of range.
• Missing or negative scenario probabilities (probabilities of leaves).
• The ancestor set contains too many entries (more than \(2^{\text{num\_nodes}}\)).
• SCENRED detects a cycle in the ancestor set.
• SCENRED detects incomplete scenarios in the ancestor set.
• Run time limit reached

6.38.9 SCENRED Errors and Error Numbers

When SCENRED encounters a serious error in the input files or in the scenario tree, it sends an error message to the screen and to the log file. These messages always start with

**** SCENRED run-time error . . .

The number of SCENRED errors are contained in the parameter \texttt{ScenRedReport} of the SCENRED output file (if it could be created). The occurrence of an error can also be detected from the last line that SCENRED sends to the screen:

**** SCENRED ErrCode= . . .

The numerical values of \texttt{ErrCode} and their meaning are given below.

<table>
<thead>
<tr>
<th>ErrCode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(for internal use)</td>
</tr>
<tr>
<td>2</td>
<td>fatal error while reading from SCENRED input file</td>
</tr>
<tr>
<td>3</td>
<td>fatal error while writing to SCENRED output file</td>
</tr>
<tr>
<td>4</td>
<td>fatal error while reading from SCENRED option file</td>
</tr>
<tr>
<td>5</td>
<td>log file cannot be opened</td>
</tr>
<tr>
<td>6</td>
<td>a memory allocation error occurred</td>
</tr>
<tr>
<td>7</td>
<td>there are missing input parameters</td>
</tr>
<tr>
<td>8</td>
<td>could not access the GAMS names for the nodes</td>
</tr>
<tr>
<td>9</td>
<td>(for internal use)</td>
</tr>
<tr>
<td>10</td>
<td>ancestor set not given or contains too many entries</td>
</tr>
<tr>
<td>11</td>
<td>node probabilities cannot be not read or are wrong</td>
</tr>
<tr>
<td>12</td>
<td>random values for the nodes cannot be read</td>
</tr>
<tr>
<td>13</td>
<td>input parameters are out of range</td>
</tr>
<tr>
<td>14</td>
<td>ancestor set contains a cycle</td>
</tr>
<tr>
<td>15</td>
<td>incomplete scenarios or forests detected</td>
</tr>
</tbody>
</table>
6.38.10 SCENRED Warnings

SCENRED warnings are caused by misspecification of the initial scenarios that can be possibly fixed. When SCENRED encounters such an error in the input files or in the scenario tree, it sends a message to the screen and to the log file. These messages always start with

**** SCENRED Warning . . .

The following list gives an overview of the cases that produce warnings, and the action taken by SCENRED in these cases.

- The user assigned an option value that is out of range.
  **Action:** Assign the default value.

- Both parameters `red_num_leaves` and `red_percentage` are assigned nontrivial values.
  **Action:** The value of `red_percentage` will be ignored.

- The scenario probabilities (probabilities of leaves) do not sum up to 1.
  **Action:** The scenario probabilities are rescaled. Assign new probabilities to the remaining (inner) nodes that are consistent with the scenario probabilities.

- Missing probabilities of inner nodes.
  **Action:** Assign node probabilities that are consistent with the scenario probabilities.

- The ancestor set contains more than one ancestor for a node.
  **Action:** SCENRED assumes to be given a successor set instead of an ancestor set (i.e., the transpose of an ancestor matrix. This means that the graph corresponding to the ancestor set has the wrong orientation). SCENRED starts the tree diagnostic for the successor set. The reduced tree will be defined in terms of a successor set as well (if the successor set passes the tree diagnostic and if SCENRED locates no fatal error during the run).

- The fast backward method delivered a result, but the result cannot be improved by the forward or backward method (running time limit reached).
  **Action:** Use the result of the fast backward method.

6.39 SCENRED2

6.39.1 Introduction

Scenred2 is a fundamental update of the well-known scenario reduction software Scenred. A lot of new features come along with the latest release version. Beside updates and extensions concerning the control of the scenario reduction action an all new device for scenario tree construction has been implemented in Scenred2. Moreover, a lot of visualization functions to plot scenario trees and scenario processes linked to the free Gnuplot plotting software are available with Scenred2 now.

**Table: Summary of basic new functions in Scenred2**

<table>
<thead>
<tr>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional options for controlling the scenario reduction</td>
<td>Scenario Reduction</td>
</tr>
<tr>
<td>New device of scenario tree construction</td>
<td>Scenario Tree Construction</td>
</tr>
<tr>
<td>Visualization of scenario trees and processes</td>
<td>Visualization</td>
</tr>
<tr>
<td>Command line interface and data export</td>
<td>Command Line Interface</td>
</tr>
</tbody>
</table>
Successful applying Scenred or Scenred2 requires a special formulation of the stochastic programming model within the Gams program. Probabilistic information must be given by a set of nodes implying a certain ancestor structure including a well-defined root node. Note that the usage of Gams/Scenred2 is basically the same as the usage of Gams/Scenred. Hence, it is recommended for new users to look at the Scenred documentation first. All details about how to organize your Gams program, how to run Scenred from the Gams program by using the gdx interface, and, of course, examples can be found in that documentation.

The Gams/Scenred2 link provides the same gdx interface. But, due to new features some small changes in controlling the options are needed. Scenred2 supports now two types of option files. The first one is the SR-Command-File which must be passed to Scenred2 together with the Scenred2 call. The second one, the SR-Option-File includes more specific options to control the selected scenario reduction or scenario construction methods and can be declared in the SR-Command-File.

### 6.39.2.1 SR-Command-File

The command file includes the basic specifications. These are input/output gdx file names, the log file name, all other file names which are needed for diverse visualization and output options. It also includes the name of the SR-Option-File.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_file</td>
<td>specify a log file name</td>
<td>yes</td>
</tr>
<tr>
<td>input_gdx</td>
<td>specify the gdx input file for Scenred</td>
<td>yes</td>
</tr>
<tr>
<td>output_gdx</td>
<td>specify the gdx output file of Scenred</td>
<td>yes</td>
</tr>
<tr>
<td>sr_option</td>
<td>specify a SR-Option-File</td>
<td>no</td>
</tr>
<tr>
<td>visual_init</td>
<td>specify a name for visualization of input tree</td>
<td>no</td>
</tr>
<tr>
<td>visual_red</td>
<td>specify a name for visualization of reduced/constructed tree</td>
<td>no</td>
</tr>
<tr>
<td>plot_scen</td>
<td>specify a name for visualization of scenario processes</td>
<td>no</td>
</tr>
<tr>
<td>out_scen</td>
<td>specify a file for scenario data output in fan format</td>
<td>no</td>
</tr>
<tr>
<td>out_tree</td>
<td>specify a file for scenario data output in tree format</td>
<td>no</td>
</tr>
</tbody>
</table>

Example:

Scenred2 must be called together with a command file, which contains at least all required options. The data exchange via the gdx interface and the Scenred2 call from the Gams program is of the form (be careful with the meanings and right order of gdx symbols):

execute_unload 'srin.gdx', ScenRedParms, n, ancestor, prob, random;
execute 'scenred2 scenred.cmd';
execute_load 'srout.gdx', ScenRedReport, ancestor=red_ancestor, prob=red_prob;

For example, the command file could be the following (note the compatible gdx file names):

```plaintext
* scenred command file 'scenred.cmd'

log_file  sr.log
input_gdx srin.gdx
output_gdx srout.gdx
sr_option scenred.opt
visual_red tree
out_scen   raw.dat
```
6.39.2.2 ScenRedParms

With the symbol list of the parameter ScenRedParms and the SR-Option-File all necessary information regarding the Scenred2 run can be assigned. The Gams parameter ScenRedParms can easily included to the Gams program by the statement:

$libInclude scenred2

Of course, the include must be stated before calling Scenred2. After that statement all supported parameters can be assigned, but at least all required parameters regarding the input scenarios. By the symbols of the parameter ScenRedParms you make also the decision of what features you exactly want to use with Scenred2. Moreover, some other useful parameters for the Scenred2 run are included in the symbol list of the parameter ScenRedParms.

Table: Supported Scenred2 parameters in ScenRedParms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_time_steps</td>
<td>path length from root to leaf</td>
<td>yes</td>
</tr>
<tr>
<td>num_leaves</td>
<td>leaves/scenarios in the initial tree</td>
<td>yes</td>
</tr>
<tr>
<td>num_nodes</td>
<td>nodes in the initial tree</td>
<td>yes</td>
</tr>
<tr>
<td>num_random</td>
<td>random variables assigned to a scenario or node</td>
<td>yes</td>
</tr>
<tr>
<td>red_num_leaves</td>
<td>desired number of preserved scenarios or leaves</td>
<td>no</td>
</tr>
<tr>
<td>red_percentage</td>
<td>desired relative distance (accuracy)</td>
<td>no</td>
</tr>
<tr>
<td>reduction_method</td>
<td>desired reduction method</td>
<td>no</td>
</tr>
<tr>
<td>construction_method</td>
<td>desired tree construction method</td>
<td>no</td>
</tr>
<tr>
<td>num_stages</td>
<td>number stages</td>
<td>no</td>
</tr>
<tr>
<td>run_time_limit</td>
<td>time limit in seconds</td>
<td>no</td>
</tr>
<tr>
<td>report_level</td>
<td>report level: more messages by higher values</td>
<td>no</td>
</tr>
<tr>
<td>scen_red</td>
<td>scenario reduction command</td>
<td>no</td>
</tr>
<tr>
<td>tree_con</td>
<td>tree construction command</td>
<td>no</td>
</tr>
<tr>
<td>visual_init</td>
<td>visualization initial tree</td>
<td>no</td>
</tr>
<tr>
<td>visual_red</td>
<td>visualization reduced (constructed) tree</td>
<td>no</td>
</tr>
<tr>
<td>plot_scen</td>
<td>visualization scenario processes</td>
<td>no</td>
</tr>
<tr>
<td>out_scen</td>
<td>output of scenario raw data</td>
<td>no</td>
</tr>
<tr>
<td>out_tree</td>
<td>output of scenario tree data</td>
<td>no</td>
</tr>
</tbody>
</table>

To enable some options assign a value to the parameter. A parameter value of zero (default) disables an option. Note that when running Scenred2 either scenario reduction or scenario tree construction can be performed. Hence, only scen_red or tree_con should be used at once.

Example:

The following statements describe a possible example setup for proceeding the scenario tree construction with visualization of the scenario tree and output of the scenarios to a raw data file afterwards. Note that for the visualization and the scenario output the name of output files must be specified in the SR-Command-File. Otherwise a warning will inform you about not selected file names.

* general parameters
ScenRedParms('num_leaves') = 100;
ScenRedParms('num_nodes') = 200;
ScenRedParms('num_time_steps') = 5;
ScenRedParms('num_random') = 2;
ScenRedParms('report_level') = 2;
ScenRedParms('run_time_limit') = 30;

* execution commands
ScenRedParms('tree_con') = 1;
ScenRedParms('visual_red') = 1;
ScenRedParms('out_scen') = 1;

Scenred2 can also be used for plotting tasks only. Disable both the scen_red and tree_con option and use one ore more visualization options only (see also Section Visualization for more details regarding visualizations).

6.39.2.3 SR-Option-File

The SR-Option-File is the more specific option file and will be passed to Scenred2 by the sr_option statement specified in the SR-Command-File. It serves as control unit for available methods provided by Scenred2. The supported options depend on what kind of method is called with Scenred2. A detailed list of all options together with examples are given below for both the scenario reduction and the scenario construction devices (see Sections Scenario Reduction and Scenario Tree Construction, respectively). Note that certain parameters can be assigned by using both ScenRedParms and the SR-Option-File. In case of having parameters defined twice a warning by Scenred2 will generated to inform you.

6.39.3 Scenario Reduction

The scenario reduction device consists of approved methods for reducing the model size by reducing the number of scenarios in an optimal way. Here it doesn't make any difference whether the input data is structured as scenario tree or not. But note, the classical scenario reduction approach is actually developed for two-stage models. Extensions for the multistage case are planed in the near future. To learn more about the mathematical theory see, for example,


With Scenred2 the most popular and accurate reduction algorithms of forward and backward type are maintained further on. New options make it possible to proceed with the scenario reduction more individual. The most important new parameter is given by the option metric_type which allows to control the reduction process by different type of probability distances. Altogether three distances can be selected (see Table below). All probability distances are associated with a special order specification which can be set by the new option order. Both options replace the old option where_random which is not used any longer.

Table: SR Options - Scenario Reduction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>red_num_leaves</td>
<td>desired number of scenarios (integer)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>red_percentage</td>
<td>relative accuracy (number from 0.0 to 1.0)</td>
</tr>
<tr>
<td>reduction_method</td>
<td>1 - Forward, 2 - Backward, 0 - Default</td>
</tr>
<tr>
<td>metric_type</td>
<td>1 - Transport (default), 2 - Fortet-Mourier, 3 - Wasserstein</td>
</tr>
<tr>
<td>p_norm</td>
<td>choice of norm (example: 0 - max, 1 - sum, 2 - Euclidian)</td>
</tr>
<tr>
<td>scaling</td>
<td>0 - scaling off, 1 - scaling on (default)</td>
</tr>
<tr>
<td>order</td>
<td>metric order (integer, default is 1)</td>
</tr>
</tbody>
</table>

Example:

For example, a valid SR-Option-File is the following:

```
* scenred option file

reduction_method 1
red_percentage 0.3
metric_type 2
order 2
p_norm 1
scaling 0
```

Lines starting with the star symbol (route symbol can be used too) provide comment lines. The star symbol can also be used to out comment and disable certain options.

6.39.4 Scenario Tree Construction

Scenario tree construction is the outstanding all new device of Scenred2. It allows to construct scenario trees as accurate input for multistage stochastic programs (cf. H. Heitsch and W. Römisch, Scenario tree modeling for multistage stochastic programs, Mathematical Programming, 118:371–406, 2009). The input are individual scenarios in form of a fan which must be allocated before calling Scenred2. A lot of options are offered to control the tree construction process. Note that in some cases due to sensibility of certain parameters some tuning is indispensable for producing good results.

Table: SR Options (basic) - Scenario Tree Construction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>construction_method</td>
<td>1 - forward, 2 - backward</td>
</tr>
<tr>
<td>reduction_method</td>
<td>1 - forward, 2 - backward, used within the iteration</td>
</tr>
<tr>
<td>first_branch</td>
<td>time period of first branch (integer)</td>
</tr>
<tr>
<td>red_percentage</td>
<td>relative accuracy (level from 0.0 to 1.0)</td>
</tr>
<tr>
<td>eps_growth</td>
<td>1 - linear, 2 - exponential</td>
</tr>
<tr>
<td>eps_evolution</td>
<td>tree structure parameter (from 0.0 to 1.0)</td>
</tr>
<tr>
<td>scaling</td>
<td>0 - scaling off, 1 - scaling on (default)</td>
</tr>
<tr>
<td>order</td>
<td>order of metric</td>
</tr>
</tbody>
</table>

The Table above displays the main options to control the tree construction process. They are very similar to the reduction options. The role of the option `red_percentage` is here to prescribe a total epsilon accuracy (level) for the approximation scheme. But the approximation scheme is based on stagewise
approximations which requires a splitting of the total level to the stages. Two strategies are offered by Scenred2 a linear and an exponential mapping of the total level to the intermediate levels. Use option \texttt{eps\_growth} to select one of them. Both strategies allow a second tuning parameter \texttt{eps\_evolution} which effects the slope of the epsilon splitting.

Even though this kind of control may generate good results for many applications, sometimes a more individual control can be needed. For example, some applications require a localization of branching stages. Moreover, to setup approximation bounds directly to stages can be very useful. To this end the standard options are extended by a new section environment.

### 6.39.4.1 Additional options - The section environment

An alternative control for the accurate constructions is provided by using the section environment. The section environment aims to establish a better monitoring of the construction process. There are overall three section types supported by Scenred2 with the same syntax.

**Branching control:**

This section environment allows to specify branching points, i.e., an explicit selection of stages serving for branching. For example, use

```
section branching
  2
  4
  6
end
```

to allow branching only at time period 2, 4, and 6. Note that each stage statement must be placed in one line. But stages can be merged. A shorter formulation of the same contents can be written in closed form

```
section branching
  2*6 2
end
```

This statement reads branching within time periods from period 2 to period 6 with increment 2 steps. Both assignments can be combined and should be used together with the \texttt{red\_percentage} option.

**Epsilon control:**

In the similar way by the epsilon section it is possible to assign epsilon tolerances for the stage approximations explicitly. This environment overcomes difficulties at times coming across with the automatic epsilon control. Note that this environment disables the option \texttt{red\_percentage}. For example, use

```
section epsilon
  2 0.04
  3*4 0.03
  5 0.02
  6 0.01
end
```
to control the approximation scheme by assigning different epsilon values per stage. Note that the value 0.03 is assigned to time period 3 and 4 in the example.

**Node control:**

The node control is the most specific control you have over the tree construction. With this environment the number of nodes of the tree which will generated can be assigned for each time stage explicitly. For example, use

```
section num_nodes
  1  1
  2*3 5
  4*5 10
  6  15
end
```

The syntax is the same as before. Note that only one section environment can be use at once. In particular, only the first section environment detected in the option file is used. The section environment can be out commented like a standard option too.

**Experimental option:**

There is one other useful option to speed up computations when building different scenario trees from exactly the same input data. In this case the scenario distances needed to compute the trees could be saved to a external file at the first run and reloaded at later runs. Hence, the distances must be calculated only once. For example, use the option

```
write_distance dist.sr2
```

to save the computed distances to the file 'dist.sr2'. To reload them at next run use the option

```
read_distance dist.sr2
```

The option is classified as experimental since no validation of the input file takes place. Before using this option, please ensure that the distances loaded with the `read_distance` option are the right ones.

**Example:**

Finally, look at the following example to see a valid SR-Option-File which can be passed to the scenario tree construction:

```
* tree construction option file

construction_method  2
reduction_method 1
order 1
scaling 0

section epsilon
  2*4   0.1
  5     0.2
  6     0.1
end
```
6.39.4.2 Example problems 'srtree.gms' and 'srpchase.gms'

Small example problems have been included to the GAMS Model Library.

The implementation can be found in the Gams programs 'srtree.gms' and 'srpchase.gms'. It might help you to practice in building scenario trees using Gams/Scenred2. The problem 'srtree.gms' converts a fan format scenario representation into a tree format representation and then reduces the tree size. The problem 'srpchase.gms' is to solve a simple stochastic purchase problem involving three stages. Sample scenarios which are generated from a fixed distribution using a random value generator serve as input for the tree construction.

6.39.5 Visualization

Visualization is another all new feature of Scenred2. In this section an easy way for making plots of scenario processes and tree structures is described. To this end you need the free Gnuplot software or any other plotting software which allows plotting directly from simple data files.

The concept of plotting tasks is the following. For each plot two files are generated, a Gnuplot access file (name.plt) and a raw data file (name.dat). The access file contains basic Gnuplot options and it can be adjusted for individual liking afterwards. The default output is the display. The supported plotting commands are

visual_init, visual_red, plot_scen

for plotting the initial tree structure, the reduced/constructed tree structure, and the scenario process(es), respectively.

Example:

For example, to visualize the constructed tree use the option

visual_red tree

within the SR-Command-File to specify the name for the output and activate the ScenRedParms parameter

ScenRedParms('visual_red') = 1;

in the Gams program. The result are the output files 'tree.plt' and 'tree.dat'. To compute the picture now you simply open the file 'tree.plt' with Gnuplot from the directory, where both output files are located (that should be the working directory). Alternatively, from the command line prompt call

>gnuplot tree.plt

Gnuplot will automatically generate the picture. Feel free to change any option in the Gnuplot access file for individual requirements. See also the Gnuplot manual for more details. In particular, to compute a well-scaled encapsulated postscript picture (eps), you simply have to uncomment a few lines in the Gnuplot option file above and to open it with Gnuplot once again.

With the command plot_scen the scenario process(es) can be visualized. Note that Scenred2 generates Gnuplot access and data files according to the number of random values.
6.39.6 Command Line Interface

The command line interface allows to run Scenred2 stand alone without using Gams. In this case the input for scenario reduction and scenario tree construction is handled by special input data files. The command file will be extended by the parameters having with the ScenRedParms otherwise.

To execute Scenred2 from the command line prompt together with a specified command file (which is required again), for example, call

```
>scenred2 command.file -nogams
```

To avoid diverse error messages do not forget the '-nogams' option to switch off the Gams interface. The command file can include some of the following options.

- `report_level <integer>`
- `runtime_limit <integer>`
- `read_scen <input file>`
- `scen_red <option file>`
- `tree_con <option file>`
- `visual_init <name>`
- `visual_red <name>`
- `plot_scen <name>`
- `out_scen <file name>`
- `out_tree <file name>`

The denotation is not accidental the same as in case of using the Gams interface. The meaning of a certain option is maintained for the command line interface. To compute any scenario reduction or scenario tree construction the same SR-Option-Files are supported. It remains to clarify the data input format which comes across with the new `read_scen` command.

6.39.6.1 Data input format

To feed Scenred2 with data the scenario parameters must be passed by the `read_scen` command. Two types of input file formats are accepted.

a) The tree format:

This file is a formatted data file including all information of the input scenarios tree. It must have a header with dimension information and the scenario data separated for each node. The header includes the type declaration, the number of nodes, and the number of random values.

The data part starts with the key word `DATA` (do not forget). The tree data has to be ordered node by node. For every node the following information is expected separated by white spaces: The unique predecessor node (root node points to itself) followed by the node probability and followed by the assigned number of random data values. All information to one node should be written to one line (only for clearness reasons). Comment lines are allowed.

Match the following conventions:

- Nodes are identified by a sequence of integer numbers.
- The root node is expected to be the node '1'.
- The predecessor of root is '1' too, i.e., the root points to itself.
• All nodes numbers require a canonical order by stages and scenarios (see example).

Example:

# input tree example for scenred

TYPE TREE

NODER 9
RANDOM 4

DATA
* PRED PROB RAND-1 RAND-2 RAND-3 RAND-4
  1 1.0 42.5 9.1 7.5 120.0
  1 1.0 39.8 11.2 8.4 90.0
  2 1.0 37.6 14.0 6.3 110.0
  3 0.5 38.9 12.4 8.1 130.0
  3 0.5 35.7 13.8 7.5 120.0
  4 0.25 40.3 14.9 7.2 120.0
  4 0.25 38.4 15.2 8.9 100.0
  5 0.3 37.6 14.9 9.3 80.0
  5 0.2 36.3 12.8 10.3 90.0
END

b) The fan format:

A scenario fan serves as input for the scenario tree construction but it can be used also for the scenario reduction. The scenario fan represents a special form of a scenario tree, where we consider individual scenarios merged to a collective root node (the root node can also be viewed here as some kind of artificial node).

Accordingly, the fan input file is a formatted data file including all information of the scenarios in individual form now. It must have a similar header with dimension information and the scenario data separated now for each scenario. The header gets the type declaration FAN instead of TREE and includes the number of scenarios, the number of time periods, and the number of random values. The data part is opened again with the DATA key word.

Every scenario is specified by a dataset including the scenario probability first followed by the different random values in ascending order w.r.t. time periods. All entries must be separated by a white space. Comment lines can be placed by the star and route symbols again. Note that in case of having an undetermined root node the mean of random values will taken for the first time period to appoint a unique root node. The example tree represented as input in scenario fan format is displayed in the next example.

Example:
# input fan example for scenred

TYPE FAN

TIME 5
SCEN 4
RANDOM 4

DATA
0.2500
42.5  9.1  7.5  120.0
39.8  11.2  8.4  90.0
37.6  14.0  6.3  110.0
38.9  12.4  8.1  130.0
40.3  14.9  7.2  120.0

0.2500
42.5  9.1  7.5  120.0
39.8  11.2  8.4  90.0
37.6  14.0  6.3  110.0
38.9  12.4  8.1  130.0
38.4  15.2  8.9  100.0

0.3000
42.5  9.1  7.5  120.0
39.8  11.2  8.4  90.0
37.6  14.0  6.3  110.0
35.7  13.8  7.5  120.0
37.6  14.9  9.3  80.0

0.2000
42.5  9.1  7.5  120.0
39.8  11.2  8.4  90.0
37.6  14.0  6.3  110.0
35.7  13.8  7.5  120.0
36.3  12.8  10.3  90.0
END

Note that even though all scenarios coincide at the first three time periods, in this example, the scenarios will represented by one node each for every time period by the fan input format. The exception is the first time period, where a unique root node is expected in general and, therefore, only one node is assigned. The following picture shows the structure of the scenario fan which is generated by the example input.
6.39 SCENRED2

6.39.6.2 Data Export

Scenred2 allows to export scenario data after computing the scenario reduction or scenario tree construction to external data files. Data export is available for both the Gams and the command line interface. To export data from Scenred2 two output options `out_tree` and `out_scen` can be used. These options generate data files according to the tree and fan format, respectively. The name of the data files will be specified in the SR-Command-File. When using the Gams interface the options must be connected by activating the corresponding ScenRedParms parameter, additionally.

6.39.7 A Simplified Interface to Scenred2: `libinclude runscenred2`

While the previously described interface between GAMS and Scenred2 provides a maximum of flexibility, it also is rather complex and error-prone. The GAMS utility `runscenred2` tries to hide most of the mechanics of the GAMS/Scenred2 interface. The call to `runscenred2` looks as follows:

```
$libInclude runscenred2 myprefix tree_con n tree p rtree rp rv1 rv2
```

Table: runscenred2 Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 myprefix</td>
<td>base name for files used with Scenred2</td>
</tr>
<tr>
<td>2 tree_con or scen_red</td>
<td>select Scenred2 action: tree construction or scenario reduction</td>
</tr>
<tr>
<td>3 n</td>
<td>the set of nodes in the tree</td>
</tr>
<tr>
<td>4 tree</td>
<td>the set of ancestor relations describing the tree</td>
</tr>
<tr>
<td>5 p</td>
<td>the parameter containing the node probabilities</td>
</tr>
<tr>
<td>6 rtree</td>
<td>the set of ancestor relations of the reduced tree (output)</td>
</tr>
<tr>
<td>7 rp</td>
<td>the parameter containing the node probabilities for the reduced tree (output)</td>
</tr>
<tr>
<td>8 rv1, rv2, ...</td>
<td>parameters containing random values of the nodes</td>
</tr>
</tbody>
</table>

The table above describes the arguments of the `runscenred2` call. Arguments 3, 4, 5, 8 and following correspond to the symbols that need to be exported to the Scenred2 data input file (done with the `execute_unload` call in the complex interface). The output arguments 6 and 7 correspond to the symbols imported from the Scenred2 data output file (done with the `execute_load` call in the complex interface). The parameters `ScenRedParms` and `ScenRedReport` are invisibly communicated with Scenred2.

The second argument instructs Scenred2 either to construct a tree (`tree_con`) or to reduce a tree (`scen_red`).

Instead of providing an explicit name for all the different files in the Scenred2 command file, the first argument determines the name of all files using a simple naming scheme. The following name scheme is observed:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Command option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sr2myprefix.log</td>
<td>log_file</td>
<td>log file name</td>
</tr>
<tr>
<td>sr2myprefix_in.gdx</td>
<td>input.gdx</td>
<td>gdx input file name</td>
</tr>
<tr>
<td>sr2myprefix_out.gdx</td>
<td>output.gdx</td>
<td>gdx output file name</td>
</tr>
<tr>
<td>sr2myprefix.opt</td>
<td>sr_option</td>
<td>option file name</td>
</tr>
<tr>
<td>sr2myprefix_vi.plt</td>
<td>visual_init</td>
<td>file name for visualization of input tree</td>
</tr>
<tr>
<td>Filename</td>
<td>Command option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>sr2myprefix_vr.plt</td>
<td>visual_red</td>
<td>file name for visualization of reduced/constructed tree</td>
</tr>
<tr>
<td>sr2myprefix_plot.plt</td>
<td>plot_scen</td>
<td>file name for visualization of scenario process</td>
</tr>
<tr>
<td>sr2myprefix_raw.dat</td>
<td>out_scen</td>
<td>file name for scenario data output in fan format</td>
</tr>
<tr>
<td>sr2myprefix_tree.dat</td>
<td>out_tree</td>
<td>file name for scenario data output in tree format</td>
</tr>
</tbody>
</table>

The first three files (log file, input.gdx and output.gdx) are always used. The only optional input file *sr_option* is read by Scenred2 if ScenRedParms('sroption')=1. When you create this file, make sure to use the proper file name. The output files are created by Scenred2 if the corresponding option is set to 1 in ScenRedParms, e.g. ScenRedParms('out_tree')=1.

In addition to a simpler communication of data between GAMS and Scenred2, the newer versions of GAMS/Scenred2 (starting with GAMS distribution 23.1) release the user of setting required fields in the ScenRedParms parameter: num_time_steps, num_leaves, num_nodes, and num_random. GAMS/Scenred2 calculates these numbers from its input data. In case the user still sets these fields, Scenred2 will ensure that the internally calculated numbers and the user provided numbers match.

### 6.40 ShellExecute

A Tool for Launching External Programs.

Author

Erwin Kalvelagen

Date

March 25, 2004

#### 6.40.1 Overview

*shellexecute* is a small wrapper program for the *ShellExecute* Windows API call. It allows you to spawn an external program based on the file type of the document to open.

GAMS provides the *execute* statement to execute external programs. In some cases we want to let the system figure out what application to start for a given document. This can be accomplished with *shellexecute*. For instance, when we call:

```plaintext
> shellexecute demo.html
```

Windows will launch the web browser and show *demo.html*. This works correctly, irrelevant whether the user installed Microsoft Internet Explorer or Netscape's web browser.
6.40.2 Usage

The command line for `shellexecute` looks like:

```
shellexecute [/verb=vvv] [/showcmd=flag] [/dir=ddd] filename args
```

The filename is either a document (e.g. `demo.html` or `book1.xls`) or an application (e.g. `winword.exe` or `notepad`). If a document is provided, the associated application will be launched. If a directory name is provided the windows explorer will be launched.

If additional parameters are specified they are considered as command line parameters for the process to be spawned.

The following options are recognized:

- `verb=vvv` Specifies the action to be performed. The allowed actions are application dependent. Some commonly available verbs include:
  
<table>
<thead>
<tr>
<th>verb</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edit</td>
<td>Launches an editor and opens the document for editing.</td>
</tr>
<tr>
<td>find</td>
<td>Initiates a search starting from the specified directory.</td>
</tr>
<tr>
<td>open</td>
<td>Launches an application. If this file is not an executable file, its associated application is launched.</td>
</tr>
<tr>
<td>print</td>
<td>Prints the document file.</td>
</tr>
<tr>
<td>properties</td>
<td>Displays the object's properties.</td>
</tr>
</tbody>
</table>

If no verb is specified the default command for the file class will be used (in many cases this is `open`).

- `showcmd=flag` Flag that specifies how an application is to be displayed when it is opened. The actual behavior is up to the launched program. The possible values are:
  
<table>
<thead>
<tr>
<th>showcmd</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_HIDE</td>
<td>Hides the window and activates another window.</td>
</tr>
<tr>
<td>SW_MAXIMIZE</td>
<td>Maximizes the specified window.</td>
</tr>
<tr>
<td>SW_MINIMIZE</td>
<td>Minimizes the specified window and activates the next top-level window in the $z$-order.</td>
</tr>
<tr>
<td>SW_RESTORE</td>
<td>Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.</td>
</tr>
<tr>
<td>SW_SHOW</td>
<td>Activates the window and displays it in its current size and position.</td>
</tr>
<tr>
<td>SW_SHOWMAXIMIZED</td>
<td>Activates the window and displays it as a maximized window.</td>
</tr>
<tr>
<td>SW_SHOWMINIMIZED</td>
<td>Activates the window and displays it as a minimized window.</td>
</tr>
<tr>
<td>SW_SHOWMINNOACTIVE</td>
<td>Displays the window as a minimized window. The active window remains active.</td>
</tr>
<tr>
<td>SW_SHOWNA</td>
<td>Displays the window in its current state. The active window remains active.</td>
</tr>
<tr>
<td>SW_SHOWNOACTIVATE</td>
<td>Displays a window in its most recent size and position. The active window remains active.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>showcmd</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SHOWNORMAL</td>
<td>Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.</td>
</tr>
</tbody>
</table>

The default is **SW_SHOWNORMAL**.

- **/dir=ddd** the default directory for the sub-process.

In many cases you will not need to use any options.

### 6.40.3 Examples

Below are some examples of `ShellExecute`. It is noted that much of the behavior is depending on the file associations that are installed on your machine. These file associations can be inspected and changed with the Windows Explorer, see Figure 1.

#### 6.40.3.1 Spawning a web browser

![Image](image1.png)

```bash
> shellexecute turkey.html
```

#### 6.40.3.2 Spawning notepad

```bash
> shellexecute trnsport.txt
```

If a different program is associated with a `.TXT` file, a different program will be launched. In my case notepad will be launched due to the file association shown in Figure 1.

#### 6.40.3.3 Spawning the GAMS IDE to view a GDX file

```bash
> shellexecute trnsport.gdx
```

#### 6.40.3.4 Spawning Excel

```bash
> shellexecute test.xls
```

#### 6.40.3.5 Calling shellexecute from GAMS

```gams
execute '=shellexecute trnsport.html';
```
6.40.3.6 A complete GAMS example

The following model is the transport model from the model library. We write the solution both to an HTML file and a CSV file. HTML is standard ASCII and is easily written using the PUT facility. CSV is even easier, as setting the FILE suffix csv.pc=5; will automatically generate comma separated values. The web browser is launched to view the HTML file, while Excel will be spawned to view the CSV file.

$title A Transportation Problem (TRNSPORT,SEQ=1)

$onText
Write a solution report in HTML and CSV and spawn a browser and Excel to view the results.

Erwin Kalvelagen, May 2004
$offText

Set
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /;

Parameter
  a(i) 'capacity of plant i in cases'
    / seattle 350
    / san-diego 600 /
  b(j) 'demand at market j in cases'
    / new-york 325
    / chicago 300
    / topeka 275 /

Table d(i,j) 'distance in thousands of miles'

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
  c(i,j) = f*d(i,j)/1000;

Variable
  x(i,j) 'shipment quantities in cases'
  z 'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
  cost 'define objective function'
  supply(i) 'observe supply limit at plant i'
  demand(j) 'satisfy demand at market j';

  cost..    z =e= sum((i,j), c(i,j)*x(i,j));

  supply(i).. sum(j, x(i,j)) =l= a(i);

  demand(j).. sum(i, x(i,j)) =g= b(j);
Model transport / all /;

solve transport using lp minimizing z;

display x.l, x.m;

*--------------------------------------------------------------------------
* write a solution report in HTML and launch the browser
*--------------------------------------------------------------------------

File html / 'results.html' /;
pseudocode html;

put 'Solution Report'/;
put 'Optimal objective:', z.l/;
put '/';

put '<table border="1" cellspacing="0" cellpadding="3">';
loop(j, put '<th>', j.tl, '</th>' );
put '</tr>'/;

loop(i, 
put '<tr><th>', i.tl, '</th>'; 
loop(j, put '<td>', x.l(i,j), '</td>' );
put '</tr>'/;
);

put '</table>'/;

putClose;

execute '=shellexecute results.html';

*--------------------------------------------------------------------------
* write a solution report in CSV and spawn Excel to view it
*--------------------------------------------------------------------------

File csv /results.csv/;
csv.pc=5;
pseudocode csv;
put "Optimal Objective",",", z.l/;
put '/';
put ' ';
loop(j, put j.tl);
put '/';

loop(i, 
  put i.tl;
  loop(j, put x.l(i,j));
  put '/';
);

putClose;
execute '=shellexecute results.csv';
6.40.3.7 Exporting to MS Access

This example will launch MS Access after exporting a parameter to a MDB file.

\$onText
Test of GDX2ACCESS. Dumps a symbol to an Access Database, and launches MS ACCESS.
\$offText

Set i / i1*i100 /;
Alias (i,j);

Parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'test.gdx',p;

execute '=gdx2access test.gdx';
execute '=ShellExecute test.mdb';

6.40.4 Common error messages

6.40.4.1 File does not exist

E:\wtools\ver000\examples>shellexecute xxxx.html
ShellExecute Version 1.0
The system cannot find the file specified

6.40.4.2 No file association

E:\wtools\ver000\examples>shellexecute table02.dbf
ShellExecute Version 1.0
No application is associated with the specified file for this operation

6.41 SQL2GMS

Author
Erwin Kalvelagen
6.41.1 Overview

**SQL2GMS** is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File (.inc) or a GAMS GDX File (.gdx).

When running the executable sql2gms.exe without command line arguments, the tool will run interactively with a built-in GUI interface. Alternatively **SQL2GMS** can be run in batch mode, which is useful when running it directly from a GAMS model without user intervention using the `$call` command at compile time or the `execute` command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

If you need to read data from an MS Access database or MS Excel spreadsheet, the accompanying tools MDB2GMS or XLS2GMS resp. GDXXRW may be more appropriate. In some cases it may be more useful to use **SQL2GMS** to read from Access and Excel data sources.

6.41.2 Requirements

**SQL2GMS** runs only on Windows PC's with MDAC (Microsoft Data Objects) installed. In many cases this will be installed on your computer. Otherwise it can be downloaded from the Microsoft Download Center (direct link to MDAC).

In order to use **SQL2GMS** effectively, you will need to have a working knowledge of SQL in order to formulate proper database queries. In addition you will need some knowledge on how to connect to your database using ODBC or ADO.

6.41.3 Batch Usage

**SQL2GMS** can be run in batch mode without user intervention from within the GAMS model by using the `$call` resp. `execute` statements or directly from command prompt while specifying all arguments in the command-line. A **SQL2GMS** batch call is of the following form:

```
sql2gms connectionString outputFile queryString
```

A proper call will at least contain the following three command-line arguments:
1. The name of the database and the instruction for connecting to the database must be specified within the connectionString, which is indicated by the option C, i.e. `C=connectionString`.

2. The name of outputFile, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option O, i.e. `O=outputFile.inc`, while the use of a GDX file is indicated by the option X, i.e. `X=outputFile.gdx`.

3. The SQL queryString, containing the SQL statement to be executed on the database, must be specified within the option Q, i.e. `Q=queryString`.

See also Command-line Arguments below for a complete list of all possible command-line arguments. Consider that the `$call` or `execute` usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the interactive built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a `@` (At sign).

Also consider to take a look at the section Strategies, mentioning some of the drawbacks of the batch usage and how to overcome them.

### 6.41.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using SQL2GMS directly from the GAMS model or command prompt.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Interpretation</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>connectionString</td>
<td>none</td>
<td>Specify the connectionString (required).</td>
</tr>
<tr>
<td>O</td>
<td>outputIncludeFile</td>
<td>none</td>
<td>Specify the name of the output file (.inc). Either <code>O=</code> or <code>X=</code> must be specified (or both).</td>
</tr>
<tr>
<td>On</td>
<td>nth outputIncludeFile</td>
<td>none</td>
<td>Match the nth query with the nth output file (.inc format) if multiple queries are used.</td>
</tr>
<tr>
<td>X</td>
<td>outputGDXFile</td>
<td>none</td>
<td>Specify the name of the output file (.gdx). Either <code>O=</code> or <code>X=</code> must be specified (or both).</td>
</tr>
<tr>
<td>Q</td>
<td>Query</td>
<td>none</td>
<td>This option can be used to specify a SQL query (required).</td>
</tr>
<tr>
<td>Qn</td>
<td>nth query</td>
<td>none</td>
<td>Match the nth query with the nth output file (.inc) format or with the nth set- or parameter name when writing to GDX if multiple queries are used.</td>
</tr>
<tr>
<td>S</td>
<td>setName</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.</td>
</tr>
<tr>
<td>Sn</td>
<td>nth setName</td>
<td>none</td>
<td>Match the nth query with the nth set in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>Y</td>
<td>setName (with expl. text)</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.</td>
</tr>
<tr>
<td>Argument</td>
<td>Interpretation</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Yn</td>
<td>nth setName (with expl. text)</td>
<td>none</td>
<td>Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>A</td>
<td>parameterName</td>
<td>none</td>
<td>If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.</td>
</tr>
<tr>
<td>An</td>
<td>nth parameterName</td>
<td>none</td>
<td>Match the nth query with the nth parameter in the GDX file if multiple queries are used.</td>
</tr>
<tr>
<td>L</td>
<td>Listing</td>
<td>disabled</td>
<td>Controls if the data is embedded in the listing file.</td>
</tr>
<tr>
<td>M</td>
<td>Mute</td>
<td>disabled</td>
<td>Controls if additional information is written to the log and include file.</td>
</tr>
<tr>
<td>B</td>
<td>Quote Blanks</td>
<td>disabled</td>
<td>Quote strings if they contain blanks or embedded quotes.</td>
</tr>
<tr>
<td>@fileName</td>
<td>ext. options file</td>
<td>none</td>
<td>Causes the program to read options from an external text file.</td>
</tr>
<tr>
<td>N</td>
<td>iniFileName</td>
<td>sql2gms.ini</td>
<td>Indicates the usage of a different INI file.</td>
</tr>
<tr>
<td>T1</td>
<td>connectionTimeOut</td>
<td>15</td>
<td>Indicates how long to wait while establishing a connection before terminating the attempt and generating an error.</td>
</tr>
<tr>
<td>T2</td>
<td>commandTimeOut</td>
<td>30</td>
<td>Indicates how long to wait while executing a command before terminating the attempt and generating an error.</td>
</tr>
<tr>
<td>T</td>
<td>timeOut</td>
<td>none</td>
<td>Sets both T1 and T2.</td>
</tr>
<tr>
<td>D</td>
<td>Debug</td>
<td>disabled</td>
<td>Generate debug information.</td>
</tr>
<tr>
<td>E</td>
<td>Empty sets</td>
<td>disabled</td>
<td>Allow an empty result set; without this option an empty result set will generate an error.</td>
</tr>
<tr>
<td>R</td>
<td>rowBatchSize</td>
<td>100</td>
<td>Row batch size; the default is 100 records.</td>
</tr>
<tr>
<td>P</td>
<td>Password</td>
<td>none</td>
<td>Specify a password for the database.</td>
</tr>
<tr>
<td>W</td>
<td>Wiring</td>
<td>none</td>
<td>Maps database columns to GAMS index positions.</td>
</tr>
</tbody>
</table>

Some more detailed remarks on the command-line arguments:

C = string (connectionString, default = none)

This option is required and specifies which database is to be used and how SQ12GMS talks to the database. A collection of valid connection strings can be found in section Connection Strings. Often the connection string will need to be surrounded by quotes, as in: C="DSN=sample". This option is demonstrated in all examples, see Example 1 - Reading a single valued Table for instance.

O = string (outputIncludeFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is
writable. UNC names can be used. An output file must be specified for batch operation: i.e. either \( O= \) or \( X= \) needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the \$include command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in Example 1 - Reading a single valued Table for instance.

\textbf{O} = \textit{string} (outputIncludeFile, default = none)

When using multiple queries in a single SQL2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

\begin{verbatim}
Q1 = "SELECT a, b FROM table"
O1 = ab.inc
Q2 = "SELECT c, d FROM table"
O2 = cd.inc
\end{verbatim}

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

\textbf{X} = \textit{string} (outputGDXFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC names can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either \( O= \) or \( X= \) needs to be specified (or both). This option is demonstrated in Example 5 - Reading Set with Explanatory Text or Example 7 - Multi-Query Batch Example for instance.

\textbf{Q} = \textit{string} (query, default = none)

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by the database we refer to the documentation that comes with your RDBMS (Relational Database Management System). The query is passed on directly to the RDBMS so the complete power and expressiveness of SQL is available including stored procedures etc. For an exact description of allowed expressions consult a text on your database system. This option is demonstrated in Example 1 - Reading a single valued Table for instance.

\textbf{Qn} = \textit{string} (query, default = none)

When using multiple queries in a single SQL2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX:

\begin{verbatim}
Q1 = "SELECT a, b FROM table"
O1 = ab.inc
Q2 = "SELECT c, d FROM table"
O2 = cd.inc
\end{verbatim}

or (GDX output file format - where several sets and parameters can be stored in a single file):
Q1="SELECT a, b FROM table"
A1=abParameter
Q2="SELECT c FROM table"
S2=cSet

Note the usage of the arguments An resp. Sn in order to store the results as parameter resp. set and to specify the name of the symbols. See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

S = string (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in Example 4 - Reading a multi dimensional Set.

Sn = string (setName, default = none)

If multiple queries are used in a single SQL2GMS call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet

See section Multi-Query Batch Usage and Example 7 - Multi-Query Batch Example for instance.

Y = string (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in Example 5 - Reading Sets with Explanatory Text for instance.

Yn = string (setName, default = none)

If multiple queries are used in a single SQL2GMS call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.

Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet

The last column specified within the select clause in the SQL statements will be used as explanatory text.

A = string (parameterName, default = none)

If we write to a GDX file, use this option to specify the name of a parameter to be stored the GDX file (containing the results of the query). Note: MDB2GMS uses P, but P was already taken in SQL2GMS for specifying the password.

An = string (parameterName, default = none)
If multiple queries are used in a single SQL2GMS call while writing to a GDX file, use this option to specify the name of the nth parameter to be stored in the GDX file (containing the results of the nth query), e.g.

\begin{verbatim}
Q1="SELECT i, j, value FROM table"
A1=ijValue
Q2="SELECT n, m, value FROM table"
A2=nmValue
\end{verbatim}

See section Multi-Query Batch Usage and Example 7 - Multi-Query Batch Example for instance.

**L** (Listing, disabled by default)

Embed the data between the $offListing and $onListing dollar control options, so the data will not be listed in the listing file. This is a quick way to reduce the size of the listing file when including very large data files into the model. Otherwise the listing file would become too large to be handled comfortably.

**M** (Mute, disabled by default)

Controls if additional information (GAMS and SQL2GMS version numbers, number of rows in the data, elapsed time, used query etc.) is written to the log and include file.

**B** (Quote Blanks, disabled by default)

Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. In the special case that the string contains both, the double quotes are replaced by single quotes. For more information see subsection Quotes. This option only applies to an output include file.

```
@fileName = string (fileName, default = none)
```

Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also Parameter Files for some further details.

```
N = string (fileName, default = none)
```

Use a different INI file than the standard sql2gms.ini located in the same directory as the executable sql2gms.exe.

```
T1 = integer (connectionTimeOut, default = 15)
```

Indicates how long to wait while establishing a connection before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for the connection to open. Default is 15. If you set the property to zero, ADO will wait indefinitely until the connection is opened. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

```
T2 = integer (commandTimeOut, default = 30)
```
Indicates how long to wait while executing a command before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for a command to execute. Default is 30. If you set the property to zero, ADO will wait indefinitely until the execution is complete. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

\[
T = \text{integer (timeOut, default = none)}
\]

Sets both \(T_1\) and \(T_2\).

D (Debug, disabled by default)

Generate debug information. This option must be specified in an INI file when using the interactive mode of SQL2GMS.

E (Empty sets, disabled by default)

Allow an empty result set; without this option an empty result set will generate an error. This option must be specified in an INI file when using the interactive mode of SQL2GMS.

\[
R = \text{integer (rowBatchSize, default = 100)}
\]

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of SQL2GMS.

\[
P = \text{string (password, default = none)}
\]

Specify a password for the database.

\[
W = \text{string (wiring, default = none)}
\]

By using the \(W\) option, one can map database columns to GAMS index positions. See model [Wiring] for reference (note that MDB2GMS is used to access the MS Access database instead of SQL2GMS, however this does not affect the wiring specification).

6.41.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the [transport] model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.accdb).

![Figure 6.57 Table: distances](image)

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM  distances
```

By running the following SQL2GMS statement, the connection to the database Sample.accdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).
The connection string is specified using the argument \texttt{C}. Note that the string is enclosed by quotes, as the string contains blanks. The database file name \texttt{Sample.accdb} is also specified within the connection string. For more information on how to connect to your database, see section \texttt{Connection Strings}. The arguments \texttt{Q} and \texttt{O} are used to specify the query and the output file name (and format).

The generated include file \texttt{distances.inc} looks like:

```plaintext
* -----------------------------------------------------
* SQL2GMS 3.0 25.2.0 r67636 ALFA Released 15Aug18 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* ADO version: 10.0
* Connection string: Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb
* Provider: MSDASQL
* Query: SELECT city1, city2, distance FROM distances
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```

Retrieving the data using SQL2GMS from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

```plaintext
Set
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc /
display d;
```

Finally, the values of the parameter \texttt{d} are displayed:

```
  new-york   chicago   topeka
seattle    2.500    1.700    1.800
san-diego  2.500    1.800    1.400
```

This example is also part of the GAMS Data Utilities Library, see model \texttt{[Distances2]} for reference. Note that the query results are written to a GDX file in addition.
6.41.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want to read the three index columns `year`, `loc`, `prod` and the value columns `sales` and `profit` from the database file `Sample.accdb`. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for `sales` resp. `profit`) and a UNION select.

Consider the table with two value columns `sales` and `profit`:

![Figure 6.58 Table: data (shortened for presentation)](image)

**Two separate Parameters**

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```sql
SELECT year, loc, prod, sales
FROM data

SELECT year, loc, prod, profit
FROM data
```

We can generate an include file `sales.inc` by running the following command:

```bash
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
```

Note that we specify the first query in order to select the sales and the relevant index columns within the `Q` argument. The query results are written to `sales.inc` using the `O` argument. Analogously we generate an include file `profit.inc` by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```bash
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
```

Retrieving the data using SQL2GMS from the database `Sample.accdb` and including the queried data in your GAMS model within the parameter declaration statements of `sales` and `profit` (at compile time) can be combined in the following way:
Set
  year 'years' / 1997*1998 /
  loc 'locations' / nyc, was, la, sfo /
  prd 'products' / hardware, software /;

$selfcall sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc, prd, sales FROM data"
$includesales.inc ;

$selfcall sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc, prd, profit FROM data"
$includeprofit.inc ;

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index type to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

SELECT year, loc, prd, 'sales', sales
FROM data
UNION
SELECT year, loc, prd, 'profit', profit
FROM data

The data is accessed, queried and written to data.inc by running the following command:

sql2gms @howToRead.txt

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT year, loc, prd, 'sales', sales FROM data UNION SELECT year, loc, prd, 'profit', profit FROM data
O=data.inc

The generated include file data.inc looks like (shortened for presentation):

* -----------------------------------------------------
* ADO version: 10.0
* Connection string: DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
* Provider: MSDASQL
* Query: SELECT year, loc, prd, 'sales', sales FROM data UNION SELECT year, loc, prd, 'profit', profit FROM data
* -----------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
Retrieving the data using SQL2GMS from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index type in order to access the data type sales resp. profit):

```gams
$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc
$offEcho
Set
  year 'years' / 1997*1998 /
  loc 'locations' / nyc, was, la, sfo /
  prd 'products' / hardware, software /
  type 'data type' / sales, profit /
$call sql2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
  $include data.inc
/;
This example is also part of the GAMS Data Utilities Library; see model [SalesProfitDB5] for reference.

6.41.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a database file. Suppose we want to read the column City1 (see table distances) in order to define the set i in the GAMS model. Make sure elements are unique by using the distinct operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```sql
SELECT distinct(City1)
FROM distances
```

The include file city1.inc looks like (header informations are removed in order to shorten the representation):

```gams
* ----------------------------------------
SAN-DIEGO
SEATTLE
* ----------------------------------------
```
All steps (data access via SQL2GMS, set definition) can be combined:

```
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT distinct(city1) FROM distances" O=city_i.inc
Set i 'canning plants' /
$include city_i.inc /
display i;
```

The display statement generates the following output in the listing file:

```
---- 56 SET i
seattle , san-diego
```

### 6.41.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the O argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing `a b c` is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. `a.b.c` will be interpreted correctly.

Depending on your DBMS (DataBase Management System), these periods must be added explicitly in a different way within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (index1, index2, index3 and dataTable are some placeholders):

```
SELECT index1, index2, index3, " " FROM dataTable
```

or by adding the periods explicitly within the select clause (`||` or `&` depending on DBMS):

```
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table `distances` stored in Sample.accdb. The following SQL2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2, '' FROM distances"
```

The include file `city_ij.inc` looks like (header informations are removed in order to shorten the representation):
Without adding the empty value field, the resulting include file would look like (shortened):

```
* -----------------------------------------------------
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
* -----------------------------------------------------
```

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

```
Set
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /;
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2, ' ' FROM distances" O=city_ij.inc
Set ij(i,j) 'two dimensional set' /
$include city_ij.inc /
display ij;
```

The display statement generates the following output in the listing file:

```
---- 75 SET ij two dimensional set
       new-york  chicago  topeka
SAN-DIEGO  YES  YES  YES
SEATTLE  YES  YES  YES
```

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the X and S resp. Y arguments, i.e. there is no need to modify the query:

```
SELECT index1, index2, index3 FROM datatable
```

when using SQL2GMS in the following way:

```
sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT index1, index2, index3 FROM datatable" X=setData.gdx S=setName
```

which will generate the file setData.gdx with a three dimensional set named `setName` containing the results of the query.
6.41.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a database file using SQL2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the Y argument in order to store the query results as a set with explanatory text in a GDX file.

![Table: setData](image)

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```sql
SELECT setElement, explText
FROM setData
```

### Writing the Query Results in an include File

The last column in the select clause will be used as explanatory text. Take in mind to add the argument B in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```gams
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" B Q="SELECT setElement, explText FROM setData" O=setData.inc
Set a /
$include setData.inc /
```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
firstSetElement "Explanatory text for the first 'set element'"
secondSetElement "Explanatory text for the second "set element"
thirdSetElement "Explanatory text for the third 'set element'"
fourthSetElement "Explanatory text for the fourth 'set element"
* -----------------------------------------------------
```

Note the handling of the quotes according to the description in B.

### Writing the Query Results in a GDX File

When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the Y argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file setData.inc:

```gams
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Y Q="SELECT setElement, explText FROM setData" X=setData.gdx
Set b;
.gdxIn setData.gdx
$load b = set_b
.gdxIn
```

Note that the name of the set in the GDX file is set\_b (specified within the Y argument), while the name of the GDX file was specified within the X argument.
6.41.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

Set i / NY, DC, LA, SF /;

Now assume a data table looks like:

<table>
<thead>
<tr>
<th>City</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>100</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>120</td>
</tr>
<tr>
<td>San Francisco</td>
<td>105</td>
</tr>
<tr>
<td>Washington DC</td>
<td>102</td>
</tr>
</tbody>
</table>

![Example Table](image)

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

6.41.3.7.1 Index Mapping done in GAMS  When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

* -----------------------------------------------------
| 'new york' 100 |
| 'los angeles' 120 |
| 'san francisco' 105 |
| 'washington dc' 102 |
* -----------------------------------------------------

Note that the single quotes are added by activating the option B (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:

Set i / NY, DC, LA, SF /;
Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;
$call sq12gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" B D="city1.inc" O="city1.inc"

Parameter dbdata(idb) /
$include city1.inc /
Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;

Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
The display statement generates the following output in the listing file:

```plaintext
---- 47 PARAMETER data
NY 100.000, DC 102.000, LA 120.000, SF 105.000
```

This example is also part of the GAMS Data Utilities Library, see model [IndexMapping3] for reference.

### 6.41.3.7.2 Index mapping done in Database

The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

![CityMapper Table](image)

This table can be used in a join to export the data in a format we can use by executing the query:

```sql
SELECT [GAMS City], value
FROM example_table, CityMapper
WHERE CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```plaintext
* -----------------------------------------------------
la 120
ny 100
sf 105
dc 102
* -----------------------------------------------------
```

All steps can be combined in the GAMS model:

```gams
Set i / NY, DC, LA, SF /;

$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_table.city
O=city2.inc
$offEcho

$call sql2gms @howToRead.txt
Parameter data(i) /
$include city2.inc
/;
display data;
```

The display statement generates the following output in the listing file:

```plaintext
----  38 PARAMETER data
NY 100.000,   DC 102.000,   LA 120.000,   SF 105.000
```

**Note:** MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model `[IndexMapping4]` for reference.

### 6.41.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual SQL2GMS execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single SQL2GMS call and write several GAMS include files containing the results of the queries, we can use the command-line arguments `Qn` and `On`. The structure of a multi-query call looks like:

```plaintext
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=sample.accdb
Q1=firstQuery
O1=outputFileName.inc
Q2=secondQuery
O2=outputFileName.inc
Q3=thirdQuery
O3=outputFileName.inc
```

The terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. We see that the argument `Qn` is matched by an argument `On`. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments `Qn`, `Sn`, `An` and `Yn`. The structure of a multi-query call when writing to a GDX file looks like:

```plaintext
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=sample.accdb
X=sample.gdx
Q1=firstQuery
S1=setName
Q2=secondQuery
S2=setName
Q3=thirdQuery
A3=parameterName
Q4=fourthQuery
A4=setName
```
Again, the terms firstQuery, secondQuery etc. are placeholders for some SQL statements. Here we see that a query Qn is matched by either a set name Sn or a parameter name An (the letter P was taken already; it is used to specify a password), i.e. the results of the first query will be stored as a set whose name is specified within the S1 argument, the results of the third query will be stored as a parameter whose name is specified within the A3 argument etc. The X argument is used to specify the name of the GDX file.

For a complete example see section Example 7 - Multi-Query Batch Example.

6.41.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file Sample.mdb):

![Table: data (shortened for presentation)](image)

We want to extract the following information:

- The set year
- The set loc
- The set prd
- The parameter sales
- The parameter profit

Output: Several include Files

This can be accomplished using the following GAMS code with multiple queries in a single SQL2GMS call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):
$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb

Q1=SELECT distinct(year) FROM data O1=year.inc
Q2=SELECT distinct(loc) FROM data O2=loc.inc
Q3=SELECT distinct(prod) FROM data O3=prod.inc
Q4=SELECT prod, loc, year, sales FROM data O4=sales.inc
Q5=SELECT prod, loc, year, profit FROM data O5=profit.inc
$offEcho
$call =sql2gms @howToRead.txt
Set y 'years' /
$include year.inc /
Set loc 'locations' /
$include loc.inc /
Set prd 'products' /
$include prod.inc /
Parameter sales(prd,loc,y) /
$include sales.inc /
display sales;
Parameter profit(prd,loc,y) /
$include profit.inc /
display profit;

This example is also part of the GAMS Data Utilities Library, see model [SalesProfitDB6] for reference.

Output: A single GDX File

The same example imported through a GDX file can look like:

$onEcho > howToRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb X=Sample.gdx

Q1=SELECT distinct(year) FROM data S1=year
Q2=SELECT distinct(loc) FROM data S2=loc
Q3=SELECT distinct(prod) FROM data
S3=prd

Q4=SELECT prod, loc, year, sales FROM data
A4=sales

Q5=SELECT prod, loc, year, profit FROM data
A5=profit

$offEcho
$call =sql2gms @howToRead.txt
$call ="%gams.sysdir%studio/studio" Sample.gdx

Set
  y 'years'
loc 'locations'
prd 'products';

Parameter
  sales(prd,loc,y)
  profit(prd,loc,y);

$gdxIn sample.gdx
$load y=year prd loc sales profit
$gdxIn

display sales, profit;

The call of the GDXViewer will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model [SalesProfitDB7] for reference.

6.41.5 Interactive Usage

When the tool is called without command-line arguments, it will startup interactively. Using it this way, one can specify options such as the connection string, the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.
• **Output GAMS Include file (*.inc)**. If you want to create a GAMS include file, then specify here the destination file. See also `outputIncludeFile` for some more detailed remarks.

![Output GAMS include file (inc)](D:\gamsmodels\data.inc)

• **Output GDX file (*.gdx)**. As an alternative to a GAMS include file, the tool can also generate a GDX file. One or both of the output files need to be specified. See also `outputGDXFile` for some more detailed notes.

![Output GAMS GDX file (gdx)](D:\gamsmodels\data.gdx)
• SQL Query. The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also Q for some more detailed notes.

![SQL query example]

• Progress Memo. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.

![Progress memo example]

• The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

• The Tables button will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.

• The Options button will pop up a window where you can specify a number of options. The connection string is an important option, which needs to be set correctly before a query can be submitted successfully.
• Pressing the **Help button** will show this documentation.

• Pressing the **OK button** will execute the query and an output file will be generated.

• The **Batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press `Ctrl-C` or choose **Edit|Paste** to copy the contents of the clipboard to a GAMS text file.

• Pressing the **Close button** will exit the application. The current settings will be saved in an INI file so when you run **SQL2GMS** again all current settings will be restored.

6.41.5.1 Options

The **Options** window can be created by pressing the options button:
The following options are available in the options window:

- **User name**: Here you can specify the user name for logging in to the RDBMS. For databases without user authentication, this can be left empty.

- **Password**: This edit box allows you to specify the password for the database system. The characters are echoed as a '∗'.

---

Figure 6.64 Options Menu of the Graphical User Interface
- **Connection String**: The connection string determines how SQL2GMS talks to the database. For more informations see Connection Strings.

  ![Connection String](image)

- **ODBC Data Sources/Drivers**: This drop down list can be used to compose a connection string when an ODBC data source or driver is involved. The list will show all configured data sources and all available ODBC drivers.

  ![ODBC Data Sources/Drivers](image)

- **Quote blanks**: Quote strings if they contain blanks or embedded quotes. See also B for some more detailed notes.

  ![Quote blanks](image)

- **Mute**: Don’t include the extra informational text (such as used query etc.) in the include file. The equivalent command-line argument is M.

  ![Mute](image)

- **No listing**: Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is L.

  ![No listing](image)

- **Format SQL**: If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

  ![Format SQL](image)
The following options are only needed in special cases:

- **NULL**: This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.

- **Time-out** values for connection time and command execution time expressed in seconds. If -1 is specified then it will use the default value which is 15 seconds for the connection and 30 for the commands. See also T1, T2 resp. T for some more detailed notes.

The following buttons have an obvious functionality:

- **OK button** will accept the changes made.

- **Cancel button** will ignore the changes made and all option settings will revert to their previous values.

- **Help button** will show this documentation.
• **Test Connection** will try to make a connection to the database using the given connection string.

If the settings are correct you will see something like:

![Test Connection](image)

The following options can only be specified in an INI file; there is no interactive equivalent:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td>Generate debug output</td>
</tr>
<tr>
<td>E</td>
<td>integer</td>
<td>Allow an empty result set; without this option an empty result set will generate an error</td>
</tr>
<tr>
<td>R</td>
<td>integer</td>
<td>Row batch size; the default is 100 records</td>
</tr>
</tbody>
</table>

### 6.41.6 Connection Strings

The connection string determines to which database the tool will try to connect. You can give simply the name of an ODBC Data Source or provide much more elaborate connection strings. Here is an example list:

- ODBC Examples
- OLE DB Examples
- MS Remote Examples

For more information consult the documentation with your database driver. ODBC drivers can be had from several sources: Microsoft delivers ODBC with a number of drivers; database providers may have likely an ODBC driver for their RDBMS available and finally there are a number of third party ODBC drivers available (e.g. from [http://www.easysoft.com](http://www.easysoft.com)).
6.41.7 ODBC Examples

In this section we show a few examples using SQL2GMS with ODBC data sources.

6.41.7.1 ODBC Driver Manager

To configure ODBC data sources use the ODBC Data Source Administrator. This tool can be invoked from the Start button menu: Settings | Control Panel, and click on the ODBC Data Sources icon:

Under Windows XP the sequence is: Control Panel | Performance and Maintenance | Administrative Tools and click on the Data Sources (ODBC) icon. Under Windows 10, access the Control Panel at first, type 'odbc' in the top right box and select either 'Set up ODBC data sources (32-bit)' or 'Set up ODBC data sources (64-bit)'. The ODBC Data Source Administrator tool looks like:

To create a new data source, click the Add button, select a driver, give it a name (this is the DSN name) and configure the data source.
6.41.7.2 Example 8 - Reading from an MS Access Database

There are several ways to export data from an SQL database into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Access Export. See also CSV Files.
2. Use the MDB2GMS tool.
3. Use SQL2GMS with a configured ODBC data source. The connection string will look like:

   "DSN=mydsn".

4. Use SQL2GMS with a DSN-less ODBC connection. In this case we need to specify both the driver and the location of the database file explicitly in the connection string. The connection string will look like:

   "Driver=Microsoft Access Driver (*.mdb, *.accdb);dbq=D:\data\mydata.accdb".

5. Use SQL2GMS with an OLE DB driver. The connection string can look like:

   "Microsoft.ACE.OLEDB.12.0;Data Source=D:\data\mydata.accdb"

6.41.7.3 Example 9 - Reading from an MS Excel Spreadsheet

There are numerous ways to export data from an Excel spreadsheet into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Excel Export.
2. Use the XLS2GMS tool.
3. Use the GDXXRW tool.
4. Use SQL2GMS with an Excel ODBC connection. An example is shown below.

Consider the spreadsheet ExcelDist.xls:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>city1</td>
<td>city2</td>
<td></td>
<td>Distance</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SEATTLE</td>
<td>NEW-YORK</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SAN-DIEGO</td>
<td>NEW-YORK</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SEATTLE</td>
<td>CHICAGO</td>
<td>1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SAN-DIEGO</td>
<td>CHICAGO</td>
<td>1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SEATTLE</td>
<td>TOPEKA</td>
<td>1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SAN-DIEGO</td>
<td>TOPEKA</td>
<td>1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After configuring a data source ExcelDist that uses the Excel ODBC driver and points to the .xls file containing the above sheet, we can use the connection string: "DSN=ExcelDist". With the database browser we see:
I.e. the table name is Sheet1$. We now can formulate the query: SELECT city1, city2, distance FROM [Sheet1$]. We need the brackets to protect the funny table name. The result is (header removed in order to shorten the presentation=:

* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* *****************************************************

Although other tools are often more convenient to use, this approach is useful if you need to select a subsection of the spreadsheet table. It is easy to select just a few columns or rows from a table using a properly formulated SQL query. The skeleton would be: SELECT columns_to_extract FROM [sheet1$] WHERE rows_to_extract.

An example of a more complex spreadsheet is (profit.xls):
A DSN-less connection string would be: "DRIVER=Microsoft Excel Driver (*.xls); DBQ=d:\gams\projects\sql2gms\ver2.0\profit.xls". The browser will show:

![Database Browser]

A possible query that maps the two value columns into a GAMS parameter is:

```
SELECT year, loc, prod, 'sales', sales
FROM [profitdata$]
UNION
SELECT year, loc, prod, 'profit', profit
FROM [profitdata$]
```

The result is (shortened for presentation):

```
* -----------------------------------------------------
1997.la.hardware.profit  8
1997.la.hardware.sales  80
1997.la.software.profit 16
1997.la.software.sales  60
1997.nyc.hardware.profit  5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit  9
1997.sfo.hardware.sales  80
1997.sfo.software.profit 10
1997.sfo.software.sales  50
1997.was.hardware.profit  7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales  70
1998.la.hardware.profit  6
1998.la.hardware.sales  70
* -----------------------------------------------------
```

This example is also part of the GAMS Data Utilities Library, see model [Excel] for reference.
6.41.7.4 Example 10 - Reading from a Text File

Microsoft delivers ODBC with a text file driver which allows you to read a text file as if it is a database table.

A fixed format file such as:

<table>
<thead>
<tr>
<th>City1</th>
<th>City2</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEATTLE</td>
<td>NEW-YORK</td>
<td>2.5</td>
</tr>
<tr>
<td>SAN-DIEGO</td>
<td>NEW-YORK</td>
<td>2.5</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>CHICAGO</td>
<td>1.7</td>
</tr>
<tr>
<td>SAN-DIEGO</td>
<td>CHICAGO</td>
<td>1.8</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>TOPEKA</td>
<td>1.8</td>
</tr>
<tr>
<td>SAN-DIEGO</td>
<td>TOPEKA</td>
<td>1.4</td>
</tr>
</tbody>
</table>

can be read using Fixed Length setting of the text driver:

The resulting include file will look like:

```
*-----------------------------------------------------
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
*-----------------------------------------------------
* ADO version: 2.7
* Connection string: DSN=text
* Query: SELECT city1, city2, distance FROM odbcdata.txt
* Provider: MSDASQL
*-----------------------------------------------------
```
Note that the text file is specified directly within the FROM clause.

A CSV file can be interpreted as a table as well, or any other separated format. We will try to read:

City1;City2;Distance
SEATTLE;NEW-YORK;2.5
SAN-DIEGO;NEW-YORK;2.5
SEATTLE;CHICAGO;1.7
SAN-DIEGO;CHICAGO;1.8
SEATTLE;TOPEKA;1.8
SAN-DIEGO;TOPEKA;1.4

This can be read using:

The actual formats used are stored by ODBC in an INI file schema.ini (located in the directory of the data files) which can be inspected directly:
6.41 SQL2GMS

This example is also part of the GAMS Data Utilities Library, see model [Text] for reference.

6.41.8 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.

- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.

- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing

we can conveniently use the save/restart facility. The individual submodel are coded as:

- Step 0 - Data Extraction: sr0.gms
$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
  i 'suppliers'
  j 'demand centers';

Parameter
  demand(j)
  supply(i)
  dist(i,j) 'distances';

$onEcho > howtoRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb
Q1=SELECT name FROM suppliers
  O1=i.inc
Q2=SELECT name FROM demandcenters
  O2=j.inc
Q3=SELECT name,demand FROM demandcenters
  O3=demand.inc
Q4=SELECT name, supply FROM suppliers
  O4=supply.inc
Q5=SELECT supplier, demandcenter, distance FROM distances
  O5=dist.inc
$offEcho

$call =sql2gms.exe @howtoRead.txt

Set i /
$include i.inc
/

Set j /
$include j.inc
/

Parameter demand /
$include demand.inc
/

Parameter supply /
$include supply.inc
/

Parameter dist /
$include dist.inc
/

display i, j, demand, supply, dist;

• Step 1 - Data Manipulation: sr1.gms
Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;

• Step 2 - Model Definition: sr2.gms

Variable
  x(i,j) 'shipment quantities in cases'
  z    'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
  ecost       'define objective function'
  esupply(i) 'observe supply limit at plant i'
  edemand(j) 'satisfy demand at market j';
ecost..    z  =e=  sum((i,j), c(i,j)*x(i,j));
esupply(i).. sum(j, x(i,j)) =l=  supply(i);
edemand(j).. sum(i, x(i,j)) =g=  demand(j);

• Step 3 - Model Solution: sr3.gms

$onText
step 3: model solution
execute as: > gams sr3 restart=s2 save=s3
$offText

option lp = cplex;

Model transport / all /;
solve transport using lp minimizing z;

• Step 4 - Report Writing: sr4.gms

$onText
step 4: report writing
execute as: > gams sr4 restart=s3
$offtext

abort$(transport.modelStat <> 1) "model not solved to optimality";
display x.l, z.l;

A model that executes all steps can be written as:
execute 'gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute 'gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute 'gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute 'gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute 'gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [SQLSr5] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

SQL2GDX1.gms

$onText
Store data from Access database into a GDX file.
$offText

execute 'gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorLevel "step 0 failed";

execute 'gdxviewer.exe trnsport.gdx';

The model [SQL2GDX1] is also part of the GAMS Data Utilities Library.

We can also let SQL2GMS create the GDX file:

SQL2GDX2.gms

$onText
Store data from Access database into a GDX file.
$offText

$onEcho > howToRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb
X=%system.fp%transportation.gdx
Q1=SELECT name FROM suppliers
S1=i
Q2=SELECT name FROM demandcenters
The first approach has the advantage that a complete audit record is available from the data moved from
the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the
database and how this was stored in the GDX file, that file gives the answer.

The model [SQL2GDX2] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

**GDXTRANSPORT.gms**

This model demonstrates how to load the transportation data from GDX file at compile time.

```gams
Set
   i 'suppliers'
   j 'demand centers';
Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';
$gdxIn transportation.gdx
$load i j demand supply dist
$gdxIn
display i, j, demand, supply, dist;
```

**DBTimestamp2.gms**

In one application I had to retrieve data from the database each morning, at the first run of the model.
The rest of the day, the data extracted that morning could be used. The following logic can implement
this:

```gams
$onText
Retrieve data from database first run each morning.
$offText
$onEcho > getdate.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb
Q=select day(now())
O=dtimestamp.inc
$offEcho
```
Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

Scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp, currentday;

if(dbtimestamp<>currentday,
   execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
   abort$errorLevel "step 0 (database access) failed";

   execute '=sql2gms.exe @getdate.txt'
);

The include file dbtimestamp.inc contains the day of the month (1...31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

The model [DBTimestamp2] is also part of the GAMS Data Utilities Library.

### 6.41.9 Parameter Files

Parameters can be specified in an external parameter file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command line as in:

```
$call =sql2gms C="DSN=sample" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command line like:

```
$call =sql2gms @"c:\My Documents\options.txt"
```

The parameter file

```
c:\My Documents\options.txt
```

can look like:

```
C=DSN=sample
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable
```
It is possible to write the parameter file from inside a GAMS model using the $echo command. The following example will illustrate this:

```gams
$set cmdfile "c:\windows\temp\commands.txt"
$echo "C=DSN=sample" > "%cmdfile%"
$echo "O=E:\models\labor.INC" >> "%cmdfile%"
$echo "Q=SELECT * FROM labor" >> "%cmdfile%"
$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

Newer versions of GAMS allow the usage of the $onEcho and $offEcho commands:

```gams
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho > "%cmdfile%"
C=DSN=sample
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho

$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

```gams
Q=SELECT prod, loc, year, 'sales', sales FROM data \n  UNION \n  SELECT prod, loc, year, 'profit', profit FROM data
```

6.41.10 Notes

6.41.10.1 ADO

ActiveX Data Objects. Microsoft's data-access object model. An object-oriented architecture for accessing data stored in SQL databases and related data sources. Accessible from a large number of host languages such as VB, C++, Delphi. Supersedes ODBC. Many SQL databases provide ADO access through either OLEDB or ODBC.
6.41.10.2 ODBC

An API and driver manager system for accessing data stored in an RDBMS. The API provides applications a way to talk to databases while the driver manager application allows users to install, configure and manage ODBC drivers for different databases.

6.41.10.3 OLEDB

Driver architecture for SQL databases. A driver is called a OLE DB provider. This is used from ADO.

6.41.10.4 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

```
\<server>\<share>\<path>\<file>
```

Examples:

```
\athlon\c\My Documents\sql2gms.rtf
```

6.41.10.5 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: `$gdxIn`, `$load`, `execute_load`, `execute_unload`. The GDX=filename command-line option will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using File|Open.

6.41.10.6 MDB2GMS

MDB2GMS is a tool to import tables from MS Access databases. This utility directly uses MS Access and DAO (Data Access Objects) resulting in a somewhat simpler interface. It is not needed to specify a connection string, but just a .acdb or .mdb file. The query mechanism is similar: a query is sent as it is to the database server and the result set is translated into a GAMS representation. For more information see MDB2GMS.

6.41.10.7 XLS2GMS

XLS2GMS is a tool to import data from an Excel spreadsheet. It considers the content of a selected range as GAMS source code. For more information XLS2GMS.
6.41.10.8 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet. For more information GDXXRW.

6.41.10.9 Quotes

Examples of handling of indices when the option B for quoting strings containing blanks is used:

6.41.10.10 $CALL Command

The $call command in GAMS will execute an external program at compile time. There are two forms:

$call externalProgram

$call =externalProgram

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form $call externalProgram which uses the command processor.

2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form $call externalProgram.

3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form $call =externalProgram. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: $call =externalProgram.

4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: $call start /u externalWindowsProgram. Otherwise, it is preferred to use: $call =externalWindowsProgram.

Attention

In general it is recommended to use the $call =externalProgram version for its better error-handling.

When command-line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2

The total length of the command-line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

$call "external program" "parameter 1" "parameter 2";
$call ="external program" "parameter 1" "parameter 2"

It is noted that the first form needs additional quotes around the whole command-line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes only if the '=' appears outside the double quotes.
6.41.10.11 Compile Time Commands

All $ commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed before an execution time command, even if it is below. As an example consider:

```plaintext
File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the $call is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```plaintext
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```plaintext
File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;
```

Notice that all $ commands do not include a semi-colon but are terminated by the end-of-line.

6.42 XLS2GMS

Author

Erwin Kalvelagen, GAMS Development Corp

Version

2.4

Date

May 2004

6.42.1 Overview

Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only GDXXRW is under active development. Therefore, we strongly recommend using GDXXRW for data exchange with Excel.

XLS2GMS is a tool to convert spreadsheet data from a Microsoft Excel spreadsheet into GAMS readable format. The source is a MS Excel spreadsheet file (*.XLS) and the target is a GAMS Include File.

When running the executable XLS2GMS.EXE without command line parameters the tool will run interactively with a built-in GUI interface. Alternatively XLS2GMS can be run in batch mode which is useful when calling it directly from a GAMS model using the $call command.

The philosophy of the tool is to consider the content of a spreadsheet as Text. This text can contain GAMS statements, or parts of GAMS statements (e.g. the data part of a table statement). The text is exported to a GAMS include file where some spacing is introduced to maintain cell boundaries. This allows tables to be exported directly to GAMS include files.
6.42 XLS2GMS

6.42.2 Requirements

XLS2GMS runs only on PC’s running Windows (95/98/NT/XP) and with MS Excel installed. Microsoft Excel is included in the MS Office suite.

6.42.3 Converting spreadsheet data to GAMS data

Spreadsheet data are often differently organized than is suitable for import into a GAMS model. In some cases the data is scattered around different sheets, and in a format that is not compatible with a more structured multi-dimensional parameter as are used in a GAMS model. To export spreadsheet data to GAMS parameters, tools will either require a strict format to be used inside the spreadsheet or they will need to offer a complex specification step where the data representation in the spreadsheet is described so that it can be understood by the tool. This tool will use the first approach: the modeler is required to lay-out the data in the spreadsheet in a well defined format. Instead of defining a new format, we use the GAMS language syntax as the required representation. In effect the spreadsheet is considered as an alternative editor for GAMS source code.

As an example consider the GAMS table in the model [TRNSPORT] which is part of the GAMS model library:

```
Table d(i,j) 'distance in thousands of miles'
     new-york  |  chicago  |   topeka
  seattle    |    2.5    |  1.7 |  1.8
  san-diego  |    2.5    |  1.8 |  1.4;
```

This table can be expressed comfortably in a spreadsheet as follows:

![Spreadsheet example](image)

XLS2GMS can convert this table into a GAMS include file, which results in:

```
* -----------------------------------------------------
* XLS2GMS Version 2.4, May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----------------------------------------------------
* Application: Microsoft Excel
* Version: 9.0
* Workbook: D:\gams projects\xls2gms\ver2.0\Book2.xls
* Sheet: Sheet1
* Range: $A$1:$D$3
* -----------------------------------------------------
  new-york  |  chicago  |   topeka
  seattle    |    2.5    |  1.7 |  1.8
  san-diego  |    2.5    |  1.8 |  1.4
* -----------------------------------------------------
```
The tool will try to keep cells in a column aligned so that table statements can be used in the GAMS model. The above file \texttt{book2.inc} can be imported directly into a GAMS model by:

\begin{verbatim}
Table d(i,j) 'distance in thousands of miles'
$include book2.inc
;
\end{verbatim}

As the tool does not expect any special formatting, we could have include the table statement in the spreadsheet, as in:

\begin{verbatim}

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
<td></td>
</tr>
</tbody>
</table>

\end{verbatim}

This would result in:

\begin{verbatim}
* -----------------------------------------------------
* XLS2GMS Version 2.4, May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----------------------------------------------------
* Application: Microsoft Excel
* Version: 9.0
* Workbook: D:\gams projects\xls2gms\ver2.0\Book3.xls
* Sheet: Sheet1
* Range: $A$1:$D$5
* -----------------------------------------------------

table d(i,j) 'distance in thousands of miles'
    new-york  seattle  san-diego
    chicago   2.5  1.7  1.8
    topeka     1.8  1.4

; ? -----------------------------------------------------
\end{verbatim}

In some cases the data will need to be copied and massaged to fit into the above format. It is convenient to add a sheet dedicated for this purpose to your workbook. This interface sheet can be filled either manually, with formulas that automatically update values, or by macro's (either recorded or programmed in VBA).

\section*{6.42.4 Importing sets}

Sets can be directly imported if they are organized vertically. The following picture shows a spreadsheet with two sets with elements \{a,b,c\} organized vertically (A1:A3) and horizontally (B5:D5).
The first set can be imported directly using:

Set i /
$call =xls2gms r=a1:a3 i=book4.xls o=set1.inc
$include set1.inc
/;

display i;

The second set is somewhat more difficult, as we need to add a separating comma between the elements. This can be accomplished by:

Set j /
$call =xls2gms r=b5:d5 s="," i=book4.xls o=set2.inc
$include set2.inc
/;

display j;

This will generate the include file:

* -----------------------------------------------------
* XLS2GMS Version 2.4, May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----------------------------------------------------
* Application: Microsoft Excel
* Version: 9.0
* Workbook: D:\gams projects\xls2gms\ver2.0\Book4.xls
* Sheet: Sheet1
* Range: $B$5:$D$5
* -----------------------------------------------------
* a,b,c
* -----------------------------------------------------

6.42.5 Importing sets and tables

The table
can be considered to contain three pieces of GAMS data:

- The set \( i \) (seattle, san-diego)
- The set \( j \) (new-york, chicago, topeka)
- The distances

All this information can be read as follows:

```gams
Set i /
$call =xls2gms r=a3:a4 i=book3.xls o=seti.inc
$include seti.inc
/
Set j /
$call =xls2gms r=b2:d2 s="," i=book3.xls o=setj.inc
$include setj.inc
/
Table d(i,j)
$call =xls2gms r=a2:d4 i=book3.xls o=pard.inc
$include pard.inc
;
display i, j, d;
```

The above $call statements can be combined onto one as follows:

```gams
$onEcho > book3a.txt
i=%system.fp%book3.xls
r1=a3:a4
o1=seti.inc
r2=b2:d2
o2=setj.inc
s2=""," r3=a2:d4
o3=pard.inc
$offEcho
$call =xls2gms @book3a.txt
Set i /
```
6.42 Multidimensional parameters

Consider the data table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>livestock yield time series (kg per head)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sheep</td>
<td>meat</td>
<td>10.6</td>
<td>11.42</td>
<td>10.6</td>
<td>9.38</td>
<td>8.97</td>
<td>6.93</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>sheep</td>
<td>milk</td>
<td>23.7</td>
<td>24.1</td>
<td>24.2</td>
<td>24.2</td>
<td>24</td>
<td>23.9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>sheep</td>
<td>wool</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sheep</td>
<td>hide</td>
<td>0.5</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>goat</td>
<td>meat</td>
<td>6.39</td>
<td>7.31</td>
<td>8.88</td>
<td>7.31</td>
<td>6.39</td>
<td>6.85</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>goat</td>
<td>milk</td>
<td>37.7</td>
<td>38.1</td>
<td>38.2</td>
<td>38.2</td>
<td>38.3</td>
<td>37.8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>goat</td>
<td>wool</td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>goat</td>
<td>hide</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>angora</td>
<td>meat</td>
<td>1.77</td>
<td>1.77</td>
<td>2.86</td>
<td>2.21</td>
<td>1.77</td>
<td>1.77</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>angora</td>
<td>milk</td>
<td>14.9</td>
<td>15.2</td>
<td>14.8</td>
<td>15.2</td>
<td>14.8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>angora</td>
<td>wool</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>angora</td>
<td>hide</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>cattle</td>
<td>milk</td>
<td>210</td>
<td>208.1</td>
<td>219.8</td>
<td>213.8</td>
<td>214.8</td>
<td>217.5</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>cattle</td>
<td>hide</td>
<td>3.3</td>
<td>3.4</td>
<td>2.9</td>
<td>3</td>
<td>2.6</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>buffalo</td>
<td>meat</td>
<td>43.73</td>
<td>45.42</td>
<td>40.61</td>
<td>37.21</td>
<td>32.2</td>
<td>32.68</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>buffalo</td>
<td>milk</td>
<td>267.1</td>
<td>269.2</td>
<td>263.8</td>
<td>219.6</td>
<td>275.5</td>
<td>285.1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>buffalo</td>
<td>hide</td>
<td>4.1</td>
<td>3.4</td>
<td>3</td>
<td>2.4</td>
<td>2.5</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>poultry</td>
<td>meat</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td>2.24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>poultry</td>
<td>egg</td>
<td>62.4</td>
<td>62.2</td>
<td>64.2</td>
<td>78.3</td>
<td>78.4</td>
<td>73.3</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In this spreadsheet the first two columns are index columns. To make this valid GAMS syntax we need to insert a dot between the index elements. A simple way is to insert a (narrow) column with a dot in each cell. This way we can import this table as:

Set
  l 'livestock types' / sheep, goat, angora, cattle, buffalo, mule, poultry /
  cl 'livestk comm' / meat, milk, wool, hide, egg /
  ty 'time periods - years' / 1974*1979 /;

$onEcho > yield.txt
I="%system.fp%yield.xls"
R=data!B2:J23
O=yield.inc
$offEcho
$call =xls2gms @yield.txt

Table yieldtl(l,cl,ty) 'livestock "yield" time series (kg per head)'
$include yield.inc
;

display yieldtl;

6.42.7 Interactive use

When the tool is called without command line parameters, it will startup interactively. Using it this way, one can specify the spreadsheet file (.XLS file), the range and the final destination file (a GAMS include file) using the built-in interactive environment. The main screen contains a number of buttons and edit boxes, which are explained below.

• **Input file (.XLS)**. This is the combo box to specify the input file. The file must be a valid MS Excel spreadsheet file (.XLS). The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\machine\directory\file.xls`.

![Input file (.xls)](image)

• **Range**. The range can be left empty in which case the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). The range can also be a name if the spreadsheet contains named ranges. The browse button will start Excel allowing you to interactively select a range.

![Range](image)

• **Output GAMS Include file (.INC)**. If you want to create a GAMS include file, then specify here the destination file. The include file will be an ASCII file that can be read by GAMS using the `$include` command. If the include file already exists, it will be overwritten.
• **Progress Memo.** This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.

```
Application: Microsoft Excel
Version: 9.0
Workbook: D:\gams projects\xls2gms\ver2\xls2gms\data
Sheet: data
Range: $B$2:$J$23
Done with D:\gams projects\xls2gms\ver2
```

• The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

• **options button** will pop up a window where you can specify a number of options.

• **help button** will show this help.

• the **OK button** is pressed the query will be executed and an include file will be generated.

• the **batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press Ctrl-C or choose Edit|Paste to copy the contents of the clipboard to a GAMS text file.

• **close button** will exit the application. The current settings will be saved in an INI file so when you run XLS2GMS again all current settings will be restored.

### 6.42.8 Options

The **Options** window can be created by pressing the options button:
The following options are available in the options window:

- **Quote blanks**: Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. (In the special case the string contains both, double quotes are replaced by single quotes). For more information see this example.

- **Mute**: Don’t include the extra informational text (such as used query etc.) in the include file.

- **No listing**: Surround the include file by $offlisting and $onlisting so that the data will not be echoed to the listing file. This can be useful for very large data files, where the listing file would become too large to be handled comfortably.

- **Separator**: This option allows you to set a separator string to be written between cell entries. By default this is a blank. In some cases it can be useful to make this a comma. See example for an example where this is used to import sets. Note: when this option is set to an empty string, the results may not be syntactically correct for GAMS. As it is difficult to see the difference between a single blank and an empty string, the user interface will give some feedback for these cases. When an empty string is used, a warning is written to the include file.

- **Range Separator**: Multiple ranges can be separated by a range separator symbol. By default this is a semi-colon. When certain non-US locales are used, the semi-colon is a list separator symbol which can be used in multi-area ranges. In case of such a conflict, it is possible to change the range separator symbol.

- **Append**: Append to the output file.

- **Browse Read-Only**: When the Browse Range button is pressed, we launch Excel and try to load the currently specified input file. If this option is checked then the input file is loaded as read-only. If this option is not checked the file is loaded normally, in which case you can change and save it.

The buttons have an obvious functionality:

- **OK button** will accept the changes made.
6.42 XLS2GMS

- **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.

- **Help button** will show this help text.

6.42.9 Batch use

When calling XLS2GMS directly from GAMS we want to specify all command and options directly from the command line or from a command file. An example is:

```
C:\tmp>xls2gms "I=c:\My Documents\test.xls" O=test.inc
```

This call will perform its task without user intervention. The batch facility can be used from inside a GAMS model, e.g.:

```
Table c(i,j) 'data from spreadsheet' / 
$call =xls2gms I=C:\tmp\test.xls O=C:\tmp\data.inc R=B1:E10 
$include C:\tmp\data.inc
/;
```

The `$call` statement is rather error prone and you will need to spend a little it of time to get the call correct and reliable.

All the possible command line options are listed in command line arguments section. A proper batch call will at least contain the following command line parameters:

1. **I=inputfilename**
2. **O=outputincludefile**

If you only specify **I=inputfilename** then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command line argument. Only if an input file and an output file is provided, the call will be considered a batch invocation.

6.42.10 Command-line Arguments
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I=inputfile</td>
<td>This option is required and specifies the name of the .XLS file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. If only the file name is used without a path, the file is searched in the current directory (this is the project directory when running under the IDE). On a network UNC names can be used, and files from another computer can be accessed, e.g. &quot;\hostname\my documents\a.xls.&quot; This option is required for batch processing.</td>
</tr>
<tr>
<td>O=outputincludefile</td>
<td>option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation.</td>
</tr>
<tr>
<td>R=range</td>
<td>range is an optional argument. If not specified the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). To specify a whole sheet use: Sheet1!. The range can also be a name if the spreadsheet contains named ranges. Both global names (e.g. R=rangename) and sheet specific range names (e.g. R=Sheet2!rangename) are recognized. To select multiple ranges in one go, you can specify: R=range1;range2;range3. This is just a short-hand for three separate invocations of xls2gms. A multiple-area range can be specified by R=area1,area2,area3. Before exporting, a new range is created consisting of the union of the areas. This can be used to drop certain rows or columns from a table.</td>
</tr>
<tr>
<td>D</td>
<td>Debug. This option can be used for debugging purposes. If specified the import filter will no run minimized but &quot;restored&quot;, i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of xls2gms.</td>
</tr>
<tr>
<td>B</td>
<td>If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off.</td>
</tr>
<tr>
<td>M</td>
<td>Run in modest or mute mode: no additional information, such as version number etc. is written to the listing file.</td>
</tr>
<tr>
<td>L</td>
<td>Embed the data in $offlisting, $onlisting. A quick way to reduce the size of the listing file.</td>
</tr>
<tr>
<td>@filename</td>
<td>Causes the program to read options from a file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if it were specified on the command line.</td>
</tr>
<tr>
<td>N=inifilename</td>
<td>Use a different Inifile than the standard xls2gms.ini located in the same directory as the executable xls2gms.exe.</td>
</tr>
<tr>
<td>A</td>
<td>Append to output files instead of overwriting them.</td>
</tr>
<tr>
<td>G=&quot;x&quot;</td>
<td>Sets the range separator symbol</td>
</tr>
<tr>
<td>S=&quot;x&quot;</td>
<td>Sets the output separator symbol</td>
</tr>
</tbody>
</table>

As invocations of xls2gms are reasonably expensive (a copy of Excel is started), there is a way to optimize related calls. From version 1.4, xls2gms allows multiple ranges to be read and multiple include files to be written in one swoop. The syntax is best explained by showing an illustrative example:

```
$call =xls2gms I=c:\tmp\x.xls R1=range1 R1=range2 R2=range3 O1=c:\tmp\f1.inc O2=c:\tmp\f2.inc
```
In this example the ranges 'range1' and 'range2' are written to the file 'f1.inc' while the range 'range3' will go to file 'f2.inc'. In general the ranges specified with Rn will be written to the file specified with On. If multiple ranges are specified, they are written sequentially to the output file.

\$call =xls2gms I=c:\tmp\x.xls R=range1 R=range2 O=c:\tmp\f1.inc

In this example the ranges 'range1' and 'range2' are written to 'f1.inc'.

6.42.11 $CALL command

The $CALL command in GAMS will execute an external program at compile time. There are two forms:

$call externalprogram

$call =externalprogram

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=', bypasses the command processor and directly executes the external program. We mention some of the differences:

- Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE.

- If you want to execute these commands you will need to use the form $call externalprogram which uses the command processor. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form $call =externalprogram.

- If it is important to stop with an appropriate error message if the external program does not exist, only use the form $call =externalprogram. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: $call =externalprogram.

- When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: $call start /w externalwindowsprogram. Otherwise, it is preferred to use: $call =externalwindowsprogram.

Attention

In general it is recommended to use the $call =externalprogram version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

$call externalprogram parameter1 parameter2

$call =externalprogram parameter1 parameter2

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

$call "$external program" "parameter 1" "parameter 2"

$call ="external program" "parameter 1" "parameter 2"

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes only if the = appears outside the double quotes.
6.42.12 Command files

Parameters can be specified in a command file. This is important if the length of the command line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command lines. Instead of specifying a long command line as in:

```
$call =xls2gms I="c:\My Documents\test.xls" O="c:\My Documents\data.inc" R="Sheet2!A1:F15"
```

we can use a command line like:

```
$call =xls2gms @"c:\My Documents\options.txt"
```

The command file `c:\My Documents\options.txt` can look like:

```
I=c:\My Documents\test.xls
O=c:\My Documents\data.inc
R=Sheet2!A1:F15
```

It is possible to write the command file from inside a GAMS model using the `$echo` command. The following example will illustrate this:

```
$set cmdfile "trnsport.txt"
$echo "I=trnsport.xls" > "%cmdfile%"
$echo "O=trnsport.inc" >> "%cmdfile%"
$call =xls2gms @"%cmdfile%"

#include trnsport.inc
```

Newer versions of GAMS allow:

```
$set cmdfile trnsport.txt
$onEcho > "%cmdfile%"
I=trnsport.xls
O=trnsport.inc
$offEcho
$call =xls2gms @"%cmdfile%"

#include trnsport.inc
```

6.42.13 Multiple-area ranges and post-processing

The following fragment is from a spreadsheet from Unesco:
Assume we want to extract the 1990 percentage distribution of current expenditure for the countries Algeria through Congo. The range to select is not a contiguous area but consists of several pieces. In Excel we can use the mouse and the Ctrl key to make a multiple selection:
### The range is \$A10:\$E10:\$G10,\$A14:\$A19,\$E14:\$G19,\$A21:\$A26,\$E21:\$G26\$, where the comma's indicate the range is a multiple-area range. In this case we have six pieces. It is important that Excel understands that the union of the pieces forms a rectangular area. If this is not the case an error will be raised. (You can check this yourself by selecting a multi-area range and copying it to a new sheet: this operation will fail if the areas together don't form a rectangle).

The extracted text file will look like:

<table>
<thead>
<tr>
<th>Country or territory</th>
<th>Teachers' emoluments as percentage of total current expenditure</th>
<th>1990</th>
<th>1996</th>
<th>1996</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Prim.</td>
<td>Sec.</td>
<td>Tert.</td>
</tr>
<tr>
<td>Algeria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angola</td>
<td></td>
<td>96.3</td>
<td>./.</td>
<td>3.7</td>
</tr>
<tr>
<td>Benin</td>
<td></td>
<td>78.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Botswana</td>
<td></td>
<td>31.1</td>
<td>48.8</td>
<td>12.2</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td></td>
<td>52.7</td>
<td>41.7</td>
<td>25.8</td>
</tr>
<tr>
<td>Burundi</td>
<td></td>
<td>46.8</td>
<td>29.1</td>
<td>22.0</td>
</tr>
<tr>
<td>Cameroon</td>
<td></td>
<td>70.5</td>
<td>./.</td>
<td>29.5</td>
</tr>
<tr>
<td>Cape Verde</td>
<td></td>
<td></td>
<td>54.7</td>
<td>17.5</td>
</tr>
<tr>
<td>Central African Republic</td>
<td></td>
<td>52.7</td>
<td>14.6</td>
<td>21.5</td>
</tr>
<tr>
<td>Chad</td>
<td></td>
<td>64.4</td>
<td>47.1</td>
<td>20.3</td>
</tr>
<tr>
<td>Comoros</td>
<td></td>
<td>70.3</td>
<td>42.4</td>
<td>20.2</td>
</tr>
<tr>
<td>Congo</td>
<td></td>
<td>83.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The range is \$A10,\$E10,\$G10,\$A14,\$E14,\$G14,\$A21,\$E21,\$G26\$, where the comma's indicate the range is a multiple-area range. In this case we have six pieces. It is important that Excel understands that the union of the pieces forms a rectangular area. If this is not the case an error will be raised. (You can check this yourself by selecting a multi-area range and copying it to a new sheet: this operation will fail if the areas together don't form a rectangle).
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s/prim\.prim /</code></td>
<td>Replace &quot;prim.&quot; by &quot;prim &quot;. We need to be careful to keep the table alignment correctly, so we replace the dot by a blank instead of just removing the dot. A dot is a special character in sed (it means &quot;any character&quot;) so we escape it by specifying \.&quot;.</td>
</tr>
<tr>
<td><code>s/Sec\.sec /</code></td>
<td>Replace &quot;Sec.&quot; by &quot;sec &quot;.</td>
</tr>
<tr>
<td><code>s/Tert\.tert /</code></td>
<td>Replace &quot;Tert.&quot; by &quot;tert &quot;.</td>
</tr>
<tr>
<td><code>s/\..\.\.\. /g</code></td>
<td>Replace &quot;...&quot; by &quot; &quot;. The dots are escaped. We add g to indicate there may be multiple instances on a line that must be replaced.</td>
</tr>
<tr>
<td><code>s/\.\./\. /g</code></td>
<td>Replace &quot;./.&quot; by &quot; &quot;. Both dots and &quot;/&quot; needs to be escaped.</td>
</tr>
</tbody>
</table>

This file is not completely suitable for using in a GAMS model. The following edits would need to be made:

1. The header labels should get rid of the trailing dot.
2. Cells with ... should be replaced by blanks.
3. Cells with ./ should be replaced by blanks.

In the GAMS distribution a subdirectory `gbin` contains lots of interesting Unix utilities. Some of these are very suited to do string processing on text files, such as `sed` and `awk`. In this case we can use `sed` with some substitution commands:

```bash
$onEcho > sedcommands.txt
I=%system.fp%unesco.xls
R=A10,E10:G10,A14:A19,E14:G19
O=unesco.inc
B
$offEcho
```

The complete GAMS formulation can look like:

```gams
$onText
New version XLS2GMS ver 2.1 can handle multiple-area ranges.
$offText

Set
c  'countries'
   / Algeria, Angola, Benin, Botswana, 'Burkina Faso', Burundi, Cameroon 'Cape Verde', 'Central African Republic', Chad, Comoros, Congo /
exp 'percentage distribution of current expenditure'
   / prim, sec, tert /;

$onEcho > commands.txt
1=%system.fp%unesco.xls
R=A10,E10:G10,A14:A19,E14:G19
O=unesco.inc
B
$offEcho

$call =xls2gms @commands.txt
$onEcho > sedcommands.txt
```
6.43 XLSDUMP

Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only GDXXRW is under active development. Therefore, we strongly recommend using GDXXRW for data exchange with Excel.

This program will write all worksheets of an Excel workbook to a gdx file. Unlike GDXXRW, the program does not require that Excel is installed. Windows platforms only.

6.43.1 Usage

XLSDump infile outfile

where

infile

A valid Excel workbook

outfile

Optional. The output gdx file. If no output file is specified, the name of the input file will be used to construct the name the output file.

6.43.2 Example

Consider the following working file 'test2.xlsx'.

Converting this workbook using xlsdump:

xlsdump test2.xlsx

Will generate the file test2.gdx. Showing this gdx file in the GAMS IDE:

The parameter VF:
6.44 XLSTALK

XLSTalk is not a GDX utility, but as the program is often used with GDXXRW, it is discussed here.

6.44.1 Usage

xlstalk <option> {-V} <file> {<parameters>}

where

<file>

Excel file name with file extension

<option>

One of the following options; see following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
<th>Parameter</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Test if Excel is running</td>
<td></td>
<td>0 Excel not running 1 Excel is running</td>
</tr>
<tr>
<td>-C</td>
<td>Close file; do not save changes.</td>
<td>&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-E</td>
<td>Test if file exists</td>
<td>&lt;file&gt;</td>
<td>0 File does not exist 1 File exists</td>
</tr>
<tr>
<td>-M</td>
<td>Status of file</td>
<td>&lt;file&gt;</td>
<td>0 Not open in Excel 1 File is open and is not modified 2 File is open and has been modified</td>
</tr>
<tr>
<td>-O</td>
<td>Open file but do not reload</td>
<td>&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-Q</td>
<td>Quit Excel if no workbooks have been modified</td>
<td>&lt;file&gt;</td>
<td>0 Excel is closed 1 Excel is still running</td>
</tr>
<tr>
<td>-R</td>
<td>Run a macro</td>
<td>&lt;file&gt; &lt;macro-name&gt; {&lt;macro-param&gt;}</td>
<td></td>
</tr>
<tr>
<td>-S</td>
<td>Save and close the file</td>
<td>&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-V</td>
<td>Verbose mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-W</td>
<td>Wait for the user to close the file</td>
<td>&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-X</td>
<td>Excel version</td>
<td></td>
<td>0 Excel is not installed 1 Excel is installed; version 2003 or earlier 2 Excel is installed; version 2007 or later</td>
</tr>
</tbody>
</table>
6.44.2 Example

In GAMS, the return value can be obtained by using 'errorlevel'.

execute 'xlstalk.exe -X';

Scalar x;
x = errorLevel;
display x;
Chapter 7

Application Programming Interfaces

The Application Programing Interfaces (APIs) to GAMS allow the seamless integration of GAMS into an application. The APIs by provides appropriate classes for the interaction with GAMS. The `GAMSDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GAMSJob` class and by using the `GAMSModelInstance` class a sequence of closely related model instances can be solved in the most efficient way. Note that the GAMS object-oriented API was not designed to offer modeling capability. The model needs to be formulated in GAMS and passed on to the `GAMSJob` class.

Five versions of object-oriented GAMS APIs:

- .NET API with Target Framework Version 4.0,
- C++ API works with C++11 or later,
- Java API works with Java SE 8 or later, and
- Python API works with Python >=3.6.
- Matlab API works with Matlab 2017b (or later) and Octave 5.2 (or later).

In addition to the object-oriented GAMS APIs, there exist an R API and expert-level (or low-level) GAMS APIs in which its usage requires advanced knowledge of GAMS component libraries.

See Executing GAMS from other Environments if you want to execute GAMS directly from an application without using APIs.

7.1 Object-oriented APIs

The Reference Manuals give an overview of the features provides by GAMS APIs, release notes, and a list of all classes with descriptions. The Tutorials provide an overview of the basic functionality of the GAMS APIs and the Examples provides a list of examples available with GAMS distribution with detailed description. See also Release Notes and Supported Platforms.
7.1.1 Reference Manuals

The GAMS APIs reference manuals give an overview of the features provided by GAMS APIs, release notes, and a list of all classes with descriptions.

- .NET API
- C++ API
- Java API
- Python API
- Matlab API

7.1.2 Tutorials

GAMS API Tutorials provide an overview of basic functionalities of GAMS APIs and allow a user to get started to work with the GAMS API.

- .NET API Tutorial
- C++ API Tutorial
- Java API Tutorial
- Python API Tutorial
- Matlab API Tutorial

7.1.3 Examples

There are several examples available for the different programming languages: .NET, C++, Java, and Python. Below we give a description of each example together with its link to the detailed page in different languages.

<table>
<thead>
<tr>
<th>Examples</th>
<th>.NET</th>
<th>C++</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>Alias</td>
<td>Alias</td>
<td>Alias</td>
<td>alias.py</td>
</tr>
<tr>
<td>Benders2Stage</td>
<td>Benders2Stage, Benders2StageMT</td>
<td>Benders2Stage, Benders2StageMT</td>
<td>Benders2Stage, Benders2StageMT</td>
<td>benders2stage.py, benders2stage_mt.py</td>
</tr>
<tr>
<td>Clad</td>
<td>Clad</td>
<td>Clad</td>
<td>Clad</td>
<td>clad.py</td>
</tr>
<tr>
<td>CutStock</td>
<td>Cutstock, SimpleCutstock</td>
<td>Cutstock-Model, SimpleCutstock, Cutstock</td>
<td>Cutstock-Class, SimpleCutstock, Cutstock</td>
<td>cutstock_class.py, simple_cutstock.py, cutstock.py</td>
</tr>
<tr>
<td>Domain Checking</td>
<td>DomainChecking, domainchecking.cpp</td>
<td>DomainChecking</td>
<td>DomainChecking</td>
<td>domain_checking.py</td>
</tr>
<tr>
<td>Graphical User Interface (GUI)</td>
<td>CutStockGUI</td>
<td>CutStockGUI</td>
<td>CutStockGUI</td>
<td>CutStockGUI</td>
</tr>
</tbody>
</table>

- CutstockGUI
### 7.1 Object-oriented APIs

<table>
<thead>
<tr>
<th>Examples</th>
<th>.NET</th>
<th>C++</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>- FarmGUI</td>
<td>FarmGUI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- InterruptGUI</td>
<td>InterruptGui</td>
<td>InterruptGUI</td>
<td>interrupt_gui.py</td>
<td></td>
</tr>
<tr>
<td>- TransportGUI</td>
<td>TransportGUI</td>
<td>TransportGUI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMS Remote Object</td>
<td>GAMSServer, GAMSClient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt</td>
<td>Interrupt</td>
<td>ConsoleInterupt</td>
<td>interrupt.py</td>
<td></td>
</tr>
<tr>
<td>Markowitz</td>
<td>Markowitz</td>
<td></td>
<td>markowitz.py</td>
<td></td>
</tr>
<tr>
<td>MessageReceiver Window</td>
<td>MessageReceiverWindow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special Values</td>
<td>SpecialValues</td>
<td>SpecialValues</td>
<td>special_values.py</td>
<td></td>
</tr>
<tr>
<td>Traveling Salesman Problem</td>
<td>Tsp</td>
<td>Tsp.py</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warehouse</td>
<td>Warehouse</td>
<td>warehouse.cpp</td>
<td>Warehouse</td>
<td>warehouse.py</td>
</tr>
<tr>
<td>Unit Test</td>
<td>NUnitTest</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These examples are available with the GAMS distribution in `[Path/To/GAMS]/apifiles/` directory. In that directory, the file `readme.txt` describes how to build and execute these examples. A GAMS script to compile and execute each example is available in `[Path/To/GAMS]/apilib_ml/` directory with a table of contents.

#### 7.1.3.1 Alias

The Object-oriented API does not have the concept of a GAMS alias. An alias cannot be entered into a GAMSDatabase by API methods. Nevertheless, if the GAMSDatabase is loaded from a GDX container (e.g. the GAMSJob.OutDB), the database can contain aliases. Such an alias can be retrieved as a GAMSSet and consists of the elements of the aliased set. The method to find out if a GAMSSet in a GAMSDatabase is indeed an alias is to check if the symbol name is different from the lookup name (i.e. `bool isAlias = db.GetSet("ii").Name != "ii"`). This example model goes through the logic of how aliases are handled in the Object-oriented API.

See Alias example in [.NET, Java, Python].

#### 7.1.3.2 Benders2Stage

This example implements a simple Benders decomposition method for a stochastic linear program. The underlying model implements a simple distribution system with stochastic demand data. This example actually consists of two programs: A sequential and a parallel implementation. In the sequential version, the master and the subproblems are implemented with the GAMSModelInstance object which allows resolving the model with modified input without regenerating the model. A GAMSModelInstance has a fixed model rim, so this provides a challenge for Benders master problem because every iteration adds new constraints (the Benders cuts) to the master. We get around this limitation of GAMSModelInstance by initializing the GAMSModelInstance of the master with a fixed number of empty (i.e. non-binding) placeholders constraints and during the run of the algorithm turn these placeholder constraints into valid Benders cuts. The parallel version extends the Benders2Stage example by solving the independent subproblems in parallel. For that we need to instantiate a separate GAMSModelInstance for each parallel
worker. We use the efficient GAMSModelInstance. CopyInstance method to accomplish this in the most effective way. The number of demand scenarios can be larger than the number of parallel workers. The distribution of work is handled through a work queue. The parallel execution of the subproblems is done in separate threads (the MT in the name of the example stands for Multi Threading), so there is very little overhead from disk activity.

See Bender2Stage example in [.NET, Java, Python] and Bender2StageMT example in [.NET, Java, Python].

7.1.3.3 Clad

This example demonstrates how to implement a complex termination criterion for a difficult MIP using GAMS/Cplex. We would like to achieve a globally optimal solution (relative gap 0%) but if solution time becomes larger than n1 seconds, we can compromise for a 10% gap, and if this is not achieved after n2 seconds, we compromise for a 20% gap, and again if this is not accomplished in n3 seconds we take whatever the solver has done so far and terminate the solve. This is implemented by executing GAMSJob.run in an independent thread and providing new tolerances for the relative gap in the main thread by supplying new GAMS/Cplex option files and triggers the processing of the new tolerance option by GAMS/Cplex through the GAMSJob.Interrupt method.

See Clad example in [.NET, Java, Python].

7.1.3.4 CutStock

This example contains two programs that implement a column generation approach to solve the cutting stock problem. In the program SimpleCutsock, the column generation scheme has been implemented in GAMS. Moreover the GAMS model with the input and output data has been wrapped in a class that separates all interaction with GAMS from the driving application. In the second example, the column generation approach has been entirely implemented in the program using GAMSJob for the master and GAMSModelInstance for the pricing problem. GAMS is used to build the master and pricing problems. The logic of the column generation method is in the application program.

See Cutstock Model example in [.NET, Java, Python], SimpleCustock example in [.NET, Java, Python], and Custock example in [.NET, Java, Python].

7.1.3.5 Domain Checking

Enforcing referential integrity also known in the GAMS lingo as domain checking is an essential and important part of GAMS. The Object-oriented API does a delayed domain checking of symbols. So you can add records to a domain controlled parameter (e.g. p(i)) even though the GAMSSet i does not contain the label (yet). The user can trigger an explicit check of the referential integrity by calling the GAMSDatabase.CheckDomains (or GAMSSymbol.CheckDomains). The Object-oriented API provides methods to access the records that violate the referential integrity (see GAMSDatabaseDomainViolation for details). Domain checking is implicitly done when the GAMSDatabase is exported to a GDX file via the GAMSDatabase.Export method or for databases provided in the GAMSJob.Run method. The implicit domain check can be suppressed (and left to GAMS when importing data) via the GAMSDatabase.SuppressAutoDomainChecking property. This example demonstrates how to trigger domain checking and how to access the records that violate the referential integrity.

See Domain checking example in [.NET, C++, Java, Python].
7.1 Object-oriented APIs

7.1.3.6 GUI Examples

Modern development frameworks like Microsoft’s Visual Studio, Python, and Java allow to quickly build graphical user interfaces. Every example consists of a simple GUI that provides data entry, executing of a GAMSJob and graphically represents the results of a GAMS model.

- **CutstockGUI**: This example allows to either load data from a database or to enter data manually. A cutting stock problem is solved by using both the GAMSJob and the GAMSModelInstance class in order to implement a decomposition approach. See also CutStock Example. See this example in [.NET].

- **InterruptGUI**: This small example demonstrates how to run a GAMS model in a graphical user interface. This has all rudimentary features we know from the GAMS IDE: starting a job, capture the GAMS log in a window, and providing a button to interrupt. The underlying mechanism to interrupt the job is similar to the Interrupt Example (GAMSJob.Interrupt), but the trigger mechanism is very different. See this example in [.NET, Java, Python].

- **FarmGUI**: For this farming model, the input data can be entered directly into several tables. The GAMS model itself is executed by using one single GAMSJob instance. Results are displayed in different charts and tables. See this example in [.NET].

- **TransportGUI**: In this example a series of transportation problems is solved using GAMSModelInstance. Data can be entered into tables or can be loaded from a database. When data is entered into a table, related tables are updated automatically. Results are shown in tables and/or a bar chart. See also Transport Model Sequence Example. See this example in [.NET, Java].

7.1.3.7 GAMS Remote Object

This example demonstrates how to implement a simple GAMS Server. The example has two parts: GAMSServer and GAMSClient. The example is configured to run the client and server on the same machine, but can easily be altered to run on different machines. Both client and server need access to the GAMSRemoteClass that implements the server. The method to communicate is very simple. The client sets up the ingredients for a GAMSJob (model text, GDX input, parameters) and sends them serialized (via a byte representation of the GDX and parameter file) to the server. The server recreates the GAMS objects from the serialized representations, runs the model, and ships GAMSJob.OutDB serialized back to the client for further processing.

See GAMSServer example in [.NET] and GAMSClient in [.NET].

7.1.3.8 Interrupt

Ctrl-C in a regular console application results in the interrupt of the entire job. GAMS users are used to the fact that Ctrl-C interrupts a solve but then continues with the execution of the remaining GAMS job. The example demonstrates how to alter some console properties to get this behavior of Ctrl-C.

See Interrupt example in [.NET, Java, Python].

7.1.3.9 Markowitz

This is a small graphical program that plots the efficient frontier of Markowitz’ portfolio selection problem with the two objectives return and risk. The example utilizes the GAMSModelInstance class to solve the parameterized objective max lambda*return - (1-lambda)*risk in the most efficient way.

See Markowitz example in [.NET, Python].
7.1.3.10 MessageReceiver Window

The little example demonstrates how to implement a custom visual log. From GAMS one can send messages to the form started by this program via the put_utility "winmsg". The form also understands some commands (a message that starts with @) to save the content of the form, to put the content into the clipboard, or to terminate the program. The MessageReceiverWindow is also distributed as an executable in the GAMS system directory ready to be used. The model mrw01 in the GAMS Test Library demonstrates the use of the program.

See MessageReceiver Window example in [.NET].

7.1.3.11 Special Values

This example shows how special values of the programming language (e.g. infinity) percolate down to GAMS. Infinity and NaN (not a number) are well defined. The GAMS Undefined and EPS need special considerations.

See Special Values example in [.NET, Java, Python].

7.1.3.12 Transport Model Sequence

This set of examples demonstrates the use of the Object-oriented API on the simple transport model transport.gms : A Transportation Problem [transport]. The detailed description of the various examples is provided in the Object-oriented API tutorials in the GAMS documentation.

See Transport Model example in [.NET, C++, Java, Python].

7.1.3.13 Traveling Salesman Problem

This example demonstrates how to use a GAMSModelInstance to implement the subtour elimination algorithm for the Traveling Salesman Problem (TSP) problem. Similar to Benders2Stage example, we have a placeholder for the subtour elimination constraint that gets generated in each iteration of the algorithm. In contrast to the Benders example, here we regenerate the GAMSModelInstance if the original number of placeholders was not big enough. We continue this process until all subtours are eliminated.

See Traveling Salesman Problem example in [.NET, Java, Python].

7.1.3.14 Unit Test

This example contains all the unit tests we run on the .Net Object-oriented API. See Unit Test example in [.NET].

7.1.3.15 Warehouse

This example demonstrates how to solve a simple GAMS model to assign stores to warehouses for different data sets in parallel. The model has been parameterized. The data can be derived from a few numbers namely the number of warehouses, stores, and some fixed cost scalar. The results of the model are written into a single result database that is protected across the parallel threads via a mutex.

See Warehouse example in [.NET, C++, Java, Python].
7.1.4 Release Notes

GAMS API Release Notes provide an overview of changes to the GAMS APIs in releases of the GAMS distribution.

- .NET API Release Notes
- Java API Release Notes
- Python API Release Notes

7.1.5 Supported Platforms

<table>
<thead>
<tr>
<th></th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAMS C++</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAMS Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAMS.NET</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAMS Python</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GAMS Matlab</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

7.2 Expert-Level APIs

In addition to the object-oriented GAMS API, there exist expert-level (or low-level) APIs to some component libraries of the GAMS system. These APIs are used internally in GAMS for a long time and help in the deployment of GAMS models. While they offer high performance and flexibility, they also require advanced knowledge and it is recommended that the object-oriented GAMS API is used if possible. The expert-level APIs are offered for C, C#, Delphi, Fortran, Java, Python, and Visual Basic.

The DCT, GEV, and GMO components are mainly used to build solver links to GAMS (see, e.g., the GAMSLinks project. In fact, all professional GAMS solver links are based on these components. They provide access to an instance of a model generated by a solve statement. If you plan to build advanced algorithms for models or need to solve a sequence of very similar models these components might represent an alternative to an implementing using the GAMS language.

While we also strive for backward compatibility with our component libraries, but programming interfaces are commonly subject to change, so programs based on these APIs might require adjustment when updating to a new GAMS version. The Release Notes have a section on these API changes.

Overviews of the exported function in pseudo code are available in the following summary pages:

- **CFG API** (GAMS Configuration Object)

- **DCT API** (GAMS Dictionary Object)

- **GDX API** (GAMS Data Exchange Object) with additional information available as PDF and CHM.
- **GEV API** (GAMS Environment Object) with additional GAMS Environment Object Options.

- **GMO API** (GAMS Modeling Object) with additional information on philosophy and design.

- **IDX API** (GAMS IDX Object)

- **OPT API** (GAMS Option Object)

- **PAL API** (GAMS Auditing and Licensing Object)

**Note**

Without having a GAMS system directory in the `PATH` environment variable, the `xyzCreateD` functions (e.g. `gdxCreateD`, `optCreateD`, ...) might cause some issues finding libraries that are only in the GAMS system directory on Windows.

### 7.2.1 Supported Platforms

<table>
<thead>
<tr>
<th></th>
<th>x86 64bit MS Windows</th>
<th>x86 64bit Linux</th>
<th>x86 64bit Mac OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C#</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delphi</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fortran*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Python</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VBA</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VB.NET</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Fortran API files are based on C code to load dynamic libraries. For Windows pure Fortran API files for Intel and Lahey Fortran compilers are included.

### 7.3 C++ API

GAMS C++ API provides a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (GAMSModelInstance).

### 7.4 Java API

GAMS Java API provides a Java programming interface to the General Algebraic Model System (GAMS). GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory
representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be
customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of
closely related model instances in a more efficient way (GAMSModelInstance).

The underlying GAMS engine relies to some extent on file based communication (e.g. the listing file)
and other unmanaged resources. The use of external resources in the Java environment requires special
attention. Hence, some objects need to be properly disposed before the Java garbage collector does its job.

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN
or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore
an anchor into the file system is required. The base object GAMSWorkspace manages the anchor to the
file system.

With the exception of GAMSWorkspace the objects in the gams namespace cannot be accessed across
different threads unless the instance is locked. The classes themselves are thread safe and multiple objects
of the class can be used from different threads (see below for restrictions on solvers that are not thread
safe within the GAMSModelInstance class).

Note

If you use multiple instances of the GAMSWorkspace in parallel, you should avoid using the same
WorkingDirectory. Otherwise you may end up with conflicting file names.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GAMSModelInstances. Some
solvers will not work in a multi-threaded application using GAMSModelInstances. For some solvers this
is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle
thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GAMSModelInstances are not available
for quadratic model types (QCP, MIQCP, RMIQCP).

This version of the GAMS Java API also does not provide support for the following GAMS components:
acronyms, GAMS compilation/execution errors, structured access to listing file, and solver options.

7.5 Python API

The gams namespace provides objects to interact with the General Algebraic Modeling System (GAMS).
Objects in this namespace allow convenient exchange of input data and model results (GamsDatabase), help
to create and run GAMS models (GamsJob), that can be customized by GAMS options (GamsOptions).
Furthermore, it introduces a way to solve a sequence of closely related model instances in the most efficient
way (GamsModelInstance).

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN
or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore
an anchor into the file system is required. The base class GamsWorkspace manages the anchor to the
file system. If external file communication is not an issue in a particular Python application, temporary
directories and files will be managed by objects in the namespace.

With the exception of GamsWorkspace the objects in the gams namespace cannot be accessed across
different threads unless the instance is locked. The classes themselves are thread safe and multiple objects
of the class can be used from different threads (see below for restrictions on solvers that are not thread
safe within the GamsModelInstance class).
Note

If you use multiple instances of the GamsWorkspace in parallel, you should avoid using the same
working directory. Otherwise you may end up with conflicting file names.

This version of the gams namespace lacks support for the following GAMS components:

Acronyms, support for GAMS compilation/execution errors (GamsJob.run just throws an exception),
structured access to listing file, and proper support for solver options.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GamsModelInstances. Some
solvers will not even work in a multi-threaded application using GamsModelInstances. For some solvers
this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle
thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GamsModelInstances are not available
for quadratic model types (QCP, MIQCP, RMIQCP).

The GAMS Python API collection contains further packages and modules for specific applications, some
of which are currently in beta status. The gams_magic API (beta) allows to integrate GAMS into
Jupyter notebooks. The API gams2numpy (beta) allows to communicate data between GAMS and numpy arrays.
gams2transfer (beta) simplifies the work of reading and writing GDX files. The used data structures allow
data to be easily converted directly to Pandas DataFrames (or other native Python data types). The
gams_engine API allows the creation of Python clients for GAMS Engine.

7.5.1 Jupyter Notebooks

Note

This feature is currently in beta status.

7.5.1.1 Introduction

GAMS Jupyter Notebooks allow to use notebook technology in combination with GAMS. If you just want
to learn GAMS there are probably better ways doing this. Notebooks allow you to combine GAMS and
Python. The former works great with well structured data and optimization models, while the latter is
very rich in features to retrieve, manipulate, and visualize data that comes in all sort of ways. Combining
GAMS and Python in a notebook it is relatively easy to tell an optimization story with text, data, graphs,
math, and models.

7.5.1.2 Getting Started

The first step in getting started with GAMS Jupyter Notebooks is to make your Python 3 installation
aware of the GAMS Python API collection described in the Getting started section of the API tutorial. We
recommend to follow the steps below which are specifically tailored for getting started with GAMS Jupyter
notebooks. While any Python >=3.7 installation is supported, we recommend the use of Anaconda or
Miniconda Python distributions.

In addition to the GAMS Python API collection, the packages jupyterlab, matplotlib, pandas, and
tabulate are required in order to enable GAMS Jupyter Notebooks and to run the provided examples
which are located in apifiles/Python/jupyter_examples. The following code section shows how to
create and set up a Miniconda Python environment for GAMS Jupyter notebooks:

Windows:
conda create -n gmsjupyter python=3.8 jupyterlab pandas tabulate matplotlib
conda activate gmsjupyter

cd C:\GAMS\33\apifiles\Python\api_38
python setup.py install (In case of permission problems run the following instead: python setup.py build)
cd C:\jupyter (or another directory that should be used for your notebooks)
jupyter notebook (this will start the jupyter notebook server)

Linux:
conda create -n gmsjupyter python=3.8 jupyterlab pandas tabulate matplotlib
conda activate gmsjupyter

cd ~/gams/gams33.1_linux_x64_64_sfx/apifiles/Python/api_38
python setup.py install (In case of permission problems run the following instead: python setup.py build)
cd ~/jupyter (or another directory that should be used for your notebooks)
jupyter notebook (this will start the jupyter notebook server)

macOS:
conda create -n gmsjupyter python=3.8 jupyterlab pandas tabulate matplotlib
conda activate gmsjupyter

python /Library/Frameworks/GAMS.framework/Resources/apifiles/Python/api_38/setup.py build -b $TMPDIR

cd ~/jupyter (or another directory that should be used for your notebooks)
jupyter notebook (this will start the jupyter notebook server)

The notebooks Millco.ipynb and Introduction.ipynb located in apifiles/Python/jupyter_examples are good starting points to get familiar with Jupyter notebooks and GAMS. The Tutorial in this documentation is derived from the Introduction.ipynb notebook.

7.5.1.3 Tutorial

GAMS Jupyter Notebooks allow to use notebook technology in combination with GAMS. If you just want to learn GAMS there are probably better ways doing this. Notebooks allow you to combine GAMS and Python. The former works great with well structured data and optimization models, while the latter is very rich in features to retrieve, manipulate, and visualize data that comes in all sort of ways. Combining GAMS and Python in a notebook it is relatively easy to tell an optimization story with text, data, graphs, math, and models.

The GAMS Jupyter Notebook builds on top of the Python 3 kernel. So by default the notebook cells are Python cells. Cells can be turned into GAMS cells, i.e. cells with GAMS syntax, using the Jupyter magic facility (first line in a cell is %%gams). GAMS magic commands enable GAMS support in Python Jupyter notebooks. Beside running GAMS code, it is possible to transfer data between GAMS and Python. In order to enable the GAMS magic commands, it is required to load the extension gams_magic:

In:
%load_ext gams_magic

There are a few other useful command in connection with running GAMS in a Jupyter notebook. Some transformation functions for pandas dataframes useful for exchange with GAMS have been collected in the DataTransform.ipynb. The next cell will execute that notebook and make such data transformation function, e.g. gt_from2dim (see below) available in this notebook. %%capture captures the output from the execution of the notebook and does not clutter your output.

In:
%%capture
%%run DataTransform.ipynb

Somehow one output from a cell is sometimes not enough, e.g. if you want to display a couple of tables. The display function allows you to do this. As an example, we display two Python lists:

In:
display([1,2,3])
display([4,5,6])

    [1, 2, 3]
    [4, 5, 6]
Running GAMS code

Running GAMS code can be done by using either \%gams (line magic) or \%\%gams (cell magic). While \%gams can be used for running a single line of GAMS code, \%\%gams makes the whole cell a GAMS cell. While \%gams can appear on any line, \%\%gams is only valid in the first line of a cell.

In:
\%gams set i;

In:
\%\%gams
set j;
parameter p(i,j);
parameter p2(i,j);

The GAMS compiler and execution system has been adjusted so one can run a GAMS cell multiple time, even if it contains a declaration or an equation definition, which is normally not possible in the GAMS system. The execution of the next two cells does not create a problem, which mimics the execution, modification, and reexecution of a cell.

In:
\%\%gams
set i / peter,paul,mary /, j / A,B,C /;
parameter p2(i,j) / set.i.set.j 1 /

In:
\%\%gams
set i / i1*i5 /, j / j1*j5 /;
parameter p2(i,j) / set.i.set.j 1 /

You won't see any output from a GAMS cell (unless there is a solve executed in the cell, see below). All output goes to the log and lst file. If you really need to see this you can use magic command \%gams log and \%gams lst to display the content of the log and listing file of the most recent GAMS execution. The next cell displays the content of listing file of the last run GAMS cell or line magic. The -e only display the section of the listing file associated with the execution:

In:
%gams display p2;
%gams_lst -e

Execution

-----   11 PARAMETER p2

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
<th>j3</th>
<th>j4</th>
<th>j5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>i2</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>i3</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>i4</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>i5</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

There is a limit to the execution, modification, and reexecution of GAMS cells. If the type or the dimensionality of a symbol changes, you will need to execute the notebook from scratch and do a controlled reset of the entire GAMS database via \%gams reset. For example, since we declared parameter p2 already over (i,j) we cannot change our mind and redefine p2 as parameter p2(i,i,j):

In:
%gams parameter p2(i,i,j);

This will give you a compilation error and an exception in the cell execution (uncomment the line in the next cell to do so):
With a `%gams_reset` we can reset the GAMS database and can declare symbols with a different type and domain/dimension. All other things in the GAMS database are gone, too. So we need to redefine the sets i and j, too. The state of the GAMS database is kept in various files that can easily clutter your directory. The `%gams_cleanup` call helps you to clean the directory of temporary files. The option `-k` keeps the most recent GAMS database, hence the `%gams_cleanup -k` is a save call anywhere in your notebook.

As mentioned above the execution of a `%gams` cell or `%gams` and `%gams_push` line magic does not produce output. If one wants to verify that the data ended up in GAMS we can display the symbols in GAMS and output the corresponding part of the listing file:

The next cell turns a Python list into a pandas.DataFrame, multiplies the value by 2 and displays the dataframe with IPythons's `display`. We actually display the transformed `p2` (via function `gt.pivot2d` found in the `DataTransformation` notebook run at the top of the notebook), so the table looks nicer. Next, we sends the pandas.DataFrame down to GAMS via the `%gams_push` command. Via the GAMS `display` and the output of the relevant part of the listing file we see that the `%gams_push` succeeded:
In:
```python
import pandas as pd
# turn the Python list p into a pandas.DataFrame p2 and send this down to GAMS
pp = pd.DataFrame(p)
# multiply the value by 2:
# display a nicer version of the dataframe:
display(gt_pivot2d(pp))
%gams parameter pp(i,j)
%gams_push pp
%gams display pp;
```

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>2.2</td>
<td>4.4</td>
</tr>
<tr>
<td>i2</td>
<td>6.6</td>
<td>8.8</td>
</tr>
<tr>
<td>i3</td>
<td>11.0</td>
<td>13.2</td>
</tr>
</tbody>
</table>

**Execution**

```plaintext
----- 18 PARAMETER pp

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>j2</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>2.200</td>
<td>4.400</td>
</tr>
<tr>
<td>i2</td>
<td>6.600</td>
<td>8.800</td>
</tr>
<tr>
<td>i3</td>
<td>11.000</td>
<td>13.200</td>
</tr>
</tbody>
</table>
```

When using numpy arrays in order to push data into GAMS, the data is assumed to be dense. The corresponding sets are defined automatically from 1..n, 1..m, etc depending on the data that is pushed.

In:
```python
import numpy as np
data = [[[1.1,-1.1], [2.2,-2.2]], [[3.3,-3.3], [4.4,-4.4]], [[5.5,-5.5],[6.6,-6.6]]
p3 = np.array(data)
```

In:
```plaintext
%gams set i, j, k; parameter p3(i,j,k);
%gams_push p3
%gams display i,j,k,p3;
%gams_lst -e
```

**Execution**

```plaintext
----- 23 SET i
1, 2, 3

----- 23 SET j
1, 2

----- 23 SET k
1, 2
```
7.5.1.3.3 Pulling Data from GAMS to Python

The line magic \%gams pull transfers data from GAMS to Python in different formats. Supported formats are lists (default), pandas.DataFrame and numpy arrays. The following example pulls the sets \textit{i}, \textit{j}, and parameter \textit{p3} from GAMS into lists. For multi-dimensional symbols the records become Python tuples. Currently, the renaming functionality \%gams_pull gamsSym=pySymbol is not yet supported.

\begin{verbatim}
In:
gams pull p3 i j
display(i,j,p3)

['1', '2', '3']
['1', '2']
[('1', '1', '1', 1.1),
 ('1', '1', '2', -1.1),
 ('1', '2', '1', 2.2),
 ('1', '2', '2', -2.2),
 ('2', '1', '1', 3.3),
 ('2', '1', '2', -3.3),
 ('2', '2', '1', 4.4),
 ('2', '2', '2', -4.4),
 ('3', '1', '1', 5.5),
 ('3', '1', '2', -5.5),
 ('3', '2', '1', 6.6),
 ('3', '2', '2', -6.6)]
\end{verbatim}

The switch \texttt{-d} will populate pandas.DataFrames instead of lists with the GAMS data. The dataframes that are pushed into or pulled from GAMS have a very specific layout. There is a record index and the GAMS domains show up as columns in the dataframe. For parameters, there is an extra \texttt{value} column. For variables and equations we find extra columns \texttt{level}, \texttt{marginal}, \texttt{lower}, \texttt{upper}, and \texttt{scale}. The method \texttt{head()} used in the IPython \texttt{display} provides only the first 5 records of a pandas.DataFrame:

\begin{verbatim}
In:
gams variable x(i) /1.1 1, 2.m 3/;
gams pull -d i j p3 x
display(i, j, p3.head(), x)
\end{verbatim}

<table>
<thead>
<tr>
<th>i</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
The data transformation functions available from `DataTransformations.ipynb` help to convert between this format and formats more suitable for display of other transformations in Python. The following lines give a quick overview of the transformation functionality:

**In:**

```gams
%gams parameter r(i,j); r(i,j) = uniformInt(1,10);
%gams pull -d r
display(r,gt_pivot2d(r),gt_from2dim(gt_pivot2d(r),['i','j','value']))
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6.0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The switch `-n` will populate numpy arrays instead lists with the GAMS parameters. This format works with parameters only! The GAMS data will be dropped into a dense numpy array:
7.5.1.3.4 Troubleshooting and Hints

- Paths to notebooks must not contain whitespaces. A notebook file itself (*.ipynb) can.
- The directory containing the notebook will clutter up with temporary files. A `%gams_cleanup` or `%gams_cleanup -k` if you want to continue working in this session.
- The temporary file are useful for debugging, see below. The naming of the temporary files is not very sophisticated, so it can come to file naming conflicts if you run two notebooks in the same directory at the same time (in different browser tabs). Create subdirectories and move the notebook into the subdirectories if you run into this problem.
- As soon as an error occurs while running GAMS code (the notebook exception is a GamsException), it can be useful to examine the listing file (*.lst) using `%cat path/to/listing.lst` or `%gams_lst`. The path of the listing file can be found in the last line of the output of a failing cell.
self.job.run(opt, checkpoint=self.cp, output=logFile)

solveSummary = self.parseTraceFile(trcFilePath)

raise gams.workspace.GamsExceptionExecution("GAMS return code not 0 (" +
else:
raise gams.workspace.GamsExceptionExecution("GAMS return code not 0 (" +
self._p = None

GamsExceptionExecution: GAMS return code not 0 (2), check /home/share/jupyter_examples/gamsJupyter16.lst

GAMS Transfer

GAMS Transfer is a package to maintain GAMS data outside a GAMS script in a programming language like Python or Matlab. It allows the user to add GAMS symbols (Sets, Aliases, Parameters, Variables and Equations), to manipulate GAMS symbols, as well as read/write symbols to different data endpoints. GAMS Transfer’s main focus is the highly efficient transfer of data between GAMS and the target programming language, while keeping those operations as simple as possible for the user. In order to achieve this, symbol records – the actual and potentially large-scale data sets – are stored in native data structures of the corresponding programming languages. The benefits of this approach are threefold: (1) The user is usually very familiar with these data structures, (2) these data structures come with a large tool box for various data operations, and (3) optimized methods for reading from and writing to GAMS can transfer the data as a bulk – resulting in the high performance of this package. This documentation describes, in detail, the use of GAMS Transfer within a Python environment.
Data within GAMS Transfer will be stored as Pandas DataFrame. The flexible nature of Pandas DataFrames makes them ideal for storing/manipulating sparse data. Pandas includes advanced operations for indexing and slicing, reshaping, merging and even visualization.

Pandas also includes a number of advanced data I/O tools that allow users to generate DataFrames directly from CSV (.csv), JSON (.json), HTML (.html), Microsoft Excel (.xls, .xlsx), SQL, pickle (.pkl), SPSS (.sav, .zsav), SAS (.xpt, .sas7bdat), etc.

Centering GAMS Transfer around the Pandas DataFrame gives GAMS users (on a variety of platforms – MacOS, Windows, Linux) access to tools to move data back and forth between their favorite environments for use in their GAMS models.

The goal of this documentation is to introduce the user to GAMS Transfer and its functionality. This documentation is not designed to teach the user how to effectively manipulate Pandas DataFrames; users seeking a deeper understanding of Pandas are referred to the extensive documentation.

7.5.2.1 Install

The user must download and install the latest version of GAMS in order to install GAMS Transfer. GAMS Transfer is installed when the GAMS Python API is built and installed. The user is referred HERE for instructions on how to install the Python API files. GAMS Transfer and all GAMS Python API files are compatible with environment managers such as Anaconda.

7.5.2.2 Design

Storing, manipulating, and transforming sparse data requires that it lives within an environment – this data can then be linked together to enable various operations. In GAMS Transfer we refer to this “environment” as the Container, it is the main repository for storing and linking our sparse data. Symbols can be added to the Container from a variety of GAMS starting points but they can also be generated directly within the Python environment using convenient function calls that are part of the GAMS Transfer package; a symbol can only belong to one container at a time.

The process of linking symbols together within a container was inspired by typical GAMS workflows but leverages aspects of object oriented programming to make linking data a natural process. Linking data enables data operations like implicit set growth, domain checking, data format transformations (to dense/sparse matrix formats), etc – all of these features are enabled by the use of ordered pandas.CategoricalDtype data types. All of these details will be discussed in the following sections.

7.5.2.2.1 Naming Conventions

Methods – functions that operate on a object – are all verbs (i.e., getMaxAbsValue(), getUniverseSet(), etc.) and use camel case for identification purposes. Methods are, by convention, tools that ”do things”; that is they involve some, potentially expensive, computations. Some GAMS Transfer methods accept arguments, while others are simply called using the () notation. Plural arguments (columns) hint that they can accept lists of inputs (i.e., a list of symbol names) while singular arguments (column) will only accept one input at a time.

Properties – inherent attributes of an object – are all nouns (i.e., name, number_records, etc.) and use snake case (lower case words separated by underscores) for identification purposes. Object properties (or “object attributes”) are fundamental to the object and therefore they are not called like methods; object properties are simply accessed by other methods or user calls. By convention, properties only require trivial amounts of computation to access.

Classes – the basic structure of an object – are all singular nouns and use camel case (starting with a capital first letter) for identification purposes.
7.5.2.3 Quick Start

7.5.2.3.1 GDX Read  
Reading in all symbols can be accomplished with one line of code (we reference data from the `trnsport.gms` example).

```python
import gamstransfer as gt
m = gt.Container("trnsport.gdx")
```
All symbol data is organized in the data attribute (a Python dict) - `m.data[<symbol_name>].records` - records are stored as Pandas DataFrames.

7.5.2.3.2 Write Symbol to CSV  
Writing symbol records to a CSV can also be accomplished with one line.

```python
m.data["x"].records.to_csv("x.csv")
```
7.5.2.3.3 Write a New GDX  
There are five symbol classes within GAMS Transfer: 1) Sets, 2) Parameters, 3) Variables, 4) Equations and 5) Aliases. For purposes of this quick start, we show how to recreate the `distance` data structure from the `trnsport.gms` model (the parameter `d`). This brief example shows how users can achieve "GAMS-like" functionality, but within a Python environment – GAMS Transfer leverages the object oriented programming to simplify syntax.

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
# create the sets i, j
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"], description="markets")
# add "d" parameter -- domain linked to set objects i and j
d = gt.Parameter(m, "d", [i, j], description="distance in thousands of miles")
# create some data as a generic DataFrame
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand miles"],
)
# setRecords will automatically convert the dist DataFrame into a standard DataFrame format
d.setRecords(dist)
# write the GDX
m.write("out.gdx")
```
This example shows a few fundamental features of GAMS Transfer:

1. An empty Container is analogous to an empty GDX file
2. Symbols will always be linked to a Container (notice that we always pass the Container reference `m` to the symbol constructor)
3. Records can be added to a symbol with the `setRecords()` method or through the `records` constructor argument (internally calls `setRecords()`). GAMS Transfer will convert many common Python data structures into a standard format.
4. Domain linking is possible by passing domain set objects to other symbols – this will also enable domain checking (violations will show up as `NaN`)
5. Writing a GDX file can be accomplished in one line with the `write()` method.

This Quick Start introduced the reader to the GAMS Transfer syntax, but in the remaining sections we will present details about the core functionality and dig further into the syntax. Specifically, we will discuss:
1. How to create a Container
2. How to add symbols to a Container
3. How to validate the data that is in the Container
4. How to defined domains implicitly with domain_forwading
5. How to describe the data that is in the Container
6. How to transform data that is in the Container
7. Understand what the universe set is (UEL list)
8. How to reorder symbols in the Container
9. How to remove symbols from the Container

### 7.5.2.4 Create a Container

The main object class within GAMS Transfer is called **Container**. The **Container** is the vessel that allows symbols to be linked together (through their domain definitions), it enables implicit set definitions, it enables structural manipulations of the data (matrix generation), and it allows the user to perform different read/write operations.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_from</td>
<td>str, GMD Object Handle, GamsDatabase Object, ConstContainer</td>
<td>Points to the source of the data being read into the Container</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>system_directory</td>
<td>str</td>
<td>Absolute path to GAMS system directory</td>
<td>No</td>
<td>Attempts to find the GAMS installation by creating a GamsWorkspace object and loading the system_directory attribute.</td>
</tr>
</tbody>
</table>

#### 7.5.2.4.1 Container constructor

Creating a **Container** is a simple matter of initializing an object. For example:

```python
import gamstransfer as gt
m = gt.Container()
```

This new **Container** object, here called `m`, contains a number of convenient properties and methods that allow the user to interact with the symbols that are in the **Container**. Some of these methods are used to filter out different types of symbols, other methods are used to numerically characterize the data within each symbol.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>main dict that is used to store all symbol data</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>

#### 7.5.2.4.2 Container properties

Symbols are organized in the **Container** under the **data** **Container** attribute. The dot notation (`m.data`) is used to access the underlying dict; symbols in this dict can then be retrieved with the standard bracket notation (`m.data[<symbol_name>]`).
In [1]: m.data
Out[1]:
{
    'i': <src.gamstransfer.Set at 0x7fa2387750a0>,
    'j': <src.gamstransfer.Set at 0x7fa238e74fa0>,
    'a': <src.gamstransfer.Parameter at 0x7fa238e74cd0>,
    'b': <src.gamstransfer.Parameter at 0x7fa238e746a0>,
    'd': <src.gamstransfer.Parameter at 0x7fa23876b370>,
    'f': <src.gamstransfer.Parameter at 0x7fa23876b400>,
    'c': <src.gamstransfer.Parameter at 0x7fa23876b5e0>,
    'x': <src.gamstransfer.Variable at 0x7fa23876b340>,
    'z': <src.gamstransfer.Variable at 0x7fa23876b640>,
    'cost': <src.gamstransfer.Equation at 0x7fa23876b2b0>,
    'supply': <src.gamstransfer.Equation at 0x7fa23876b310>,
    'demand': <src.gamstransfer.Equation at 0x7fa23876b460>
}

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>addAlias</td>
<td>Container method to add an Alias</td>
<td>name (str) alias_with (Set, Alias)</td>
<td>Alias object</td>
</tr>
<tr>
<td>addEquation</td>
<td>Container method to add an Equation</td>
<td>name (str) type (str) domain=[] (str/list) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description=&quot;&quot; (str)</td>
<td>Equation object</td>
</tr>
<tr>
<td>addParameter</td>
<td>Container method to add a Parameter</td>
<td>name (str) domain=None (str/list,None) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description=&quot;&quot; (str)</td>
<td>Parameter object</td>
</tr>
<tr>
<td>addSet</td>
<td>Container method to add a Set</td>
<td>name (str) domain=None (str/list,None) is_singleton=False (bool) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description=&quot;&quot; (str)</td>
<td>Set object</td>
</tr>
<tr>
<td>addVariable</td>
<td>Container method to add an Variable</td>
<td>name (str) type=&quot;free&quot; (str) domain=[] (str/list) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description=&quot;&quot; (str)</td>
<td>Variable object</td>
</tr>
<tr>
<td>describeAliases</td>
<td>create a summary table with descriptive statistics for Aliases</td>
<td>symbols=None (None,str/list) - if None, assumes all aliases</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------</td>
<td>--------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>describeParameters</td>
<td>create a summary table with descriptive statistics for Parameters</td>
<td>symbols=None (None,str,list) - if None, assumes all parameters</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describEquations</td>
<td>create a summary table with descriptive statistics for Equations</td>
<td>symbols=None (None,str,list) - if None, assumes all equations</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describeSets</td>
<td>create a summary table with descriptive statistics for Sets</td>
<td>symbols=None (None,str,list) - if None, assumes all sets</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describeVariables</td>
<td>create a summary table with descriptive statistics for Variables</td>
<td>symbols=None (None,str,list) - if None, assumes all variables</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>get the count of how many records contain at least one domain violation for all symbols in the Container</td>
<td>-</td>
<td>dict</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records that have domain violations for all symbols in the Container</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are any symbols with duplicate records, False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist.</td>
<td>-</td>
<td>dict</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from all symbols in the Container – keep argument can take values of ”first” (keeps the first instance of a duplicate record), ”last” (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>getSymbols</td>
<td>returns a list of object references for symbols</td>
<td>symbols (str,list)</td>
<td>list</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>getUniverseSet</td>
<td>provides a universe for all symbols, the symbols argument allows GAMS Transfer to create a partial universe if writing only a subset of symbols (currently only supported when writing to GamsDatabases or GMD Objects)</td>
<td>symbols=None (None,str,list)</td>
<td>list</td>
</tr>
<tr>
<td>isValid</td>
<td>True if all symbols in the Container are valid</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>listAliases</td>
<td>list all aliases (is_valid=None), list all valid aliases (is_valid=True), list all invalid aliases (is_valid=False) in the container</td>
<td>is_valid=None (bool,None)</td>
<td>list</td>
</tr>
<tr>
<td>listEquations</td>
<td>list all equations (is_valid=None), list all valid equations (is_valid=True), list all invalid equations (is_valid=False) in the container</td>
<td>is_valid=None (bool,None) types=None (list of equation types) - if None, assumes all types</td>
<td>list</td>
</tr>
<tr>
<td>listParameters</td>
<td>list all parameters (is_valid=None), list all valid parameters (is_valid=True), list all invalid parameters (is_valid=False) in the container</td>
<td>is_valid=None (bool,None)</td>
<td>list</td>
</tr>
<tr>
<td>listSets</td>
<td>list all sets (is_valid=None), list all valid sets (is_valid=True), list all invalid sets (is_valid=False) in the container</td>
<td>is_valid=None (bool,None)</td>
<td>list</td>
</tr>
<tr>
<td>listSymbols</td>
<td>list all symbols (is_valid=None), list all valid symbols (is_valid=True), list all invalid symbols (is_valid=False) in the container</td>
<td>is_valid=None (bool,None)</td>
<td>list</td>
</tr>
<tr>
<td>listVariables</td>
<td>list all variables (is_valid=None), list all valid variables (is_valid=True), list all invalid variables (is_valid=False) in the container</td>
<td>is_valid=None (bool,None) types=None (list of variable types) - if None, assumes all types</td>
<td>list</td>
</tr>
</tbody>
</table>
### 7.5 Python API

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>main method to read load_from, can be provided with a list of symbols to read in subsets, records controls if symbol records are loaded or just metadata</td>
<td><code>load_from</code> (str,GMD_Object Handle,GamsDatabase_Object,ConstContainer) symbols=&quot;all&quot; (str, list) records=True (bool)</td>
<td>None</td>
</tr>
<tr>
<td>removeSymbols</td>
<td>symbols to remove from the Container, also sets the symbols ref_container to None</td>
<td><code>symbols</code> (str,list)</td>
<td>None</td>
</tr>
<tr>
<td>renameSymbol</td>
<td>rename a symbol in the Container</td>
<td><code>old_name</code> (str), <code>new_name</code> (str)</td>
<td>None</td>
</tr>
<tr>
<td>reorderSymbols</td>
<td>reorder symbols in order to avoid domain violations</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>write</td>
<td>main bulk write method to a write_to target</td>
<td><code>write_to</code> (str,GamsDatabase,GMD_Object) <code>write_symbols=None</code> (None,str,list) - if None, assumes all symbols compress=False (bool) <code>usel_priority=None</code> (str,list) <code>merge_symbols=None</code> (None,str,list)</td>
<td>None</td>
</tr>
</tbody>
</table>

### 7.5.2.4.3 Container methods

#### 7.5.2.5 Create a Set

There are two different ways to create a GAMS set and add it to a Container.

1. Use `Set` constructor

2. Use the `Container` method `addSet` (which internally calls the `Set` constructor)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>List of domains given either as string ('*' for universe set) or as reference to a Set/Alias object</td>
<td>No</td>
<td>['**']</td>
</tr>
<tr>
<td>is_singleton</td>
<td>bool</td>
<td>Indicates if set is a singleton set (True) or not (False)</td>
<td>No</td>
<td>False</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>bool</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>False</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>
### 7.5.2.5.1 Set Constructor

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;\ast]\ \ast\ n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain</td>
<td>list of domains given either as string (* for universe set) or as reference to the Set/Alias object</td>
<td>list</td>
<td>-</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>is_singleton</td>
<td>bool if symbol is a singleton set</td>
<td>bool</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns len(self.records) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>ref_container</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
<td>-</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>

### 7.5.2.5.2 Set Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>returns the count of how many records contain at least one domain violation</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records from the symbol that contain a domain violation</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are duplicate records in the symbol, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from the symbol – keep argument can take values of &quot;first&quot; (keeps the first instance of a duplicate record), &quot;last&quot; (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>findDuplicateRecords</td>
<td>get the index of records that contain any duplicate domains – keep argument can take values of &quot;first&quot; (finds all duplicates while keeping the first instance as unique), &quot;last&quot; (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)</td>
<td>keep=&quot;first&quot;</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=True, recheck a symbol if force=True</td>
<td>verbose=False force=True</td>
<td>bool</td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard pandas.DataFrame formatted records</td>
<td>records (many types)</td>
<td>None</td>
</tr>
</tbody>
</table>

### 7.5.2.5.3 Set Methods

### 7.5.2.5.4 Adding Set Records  
Three possibilities exist to assign symbol records to a set (roughly ordered in complexity):

1. Setting the argument records in the set constructor/container method (internally calls setRecords) - creates a data copy
2. Using the symbol method setRecords - creates a data copy
3. Setting the property records directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the set constructor. This is an optional keyword argument and internally the set constructor will simply call the setRecords method. The symbol method setRecords is a convenience method that transforms the given data into an approved Pandas DataFrame format (see GAMS Transfer Standard Data Formats). Many native python data types can be easily transformed into DataFrames, so the setRecords method for Set objects will accept a number of different types for input. The setRecords method is called internally on any data structure that is passed through the records argument. We show a few examples of ways to create differently structured sets:

Example #1 - Create a 1D set from a list

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
# NOTE: the above syntax is equivalent to -
i1 = gt.Set(m, "i")
# i.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
m.addSet("i", records=["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
i1 = m.addSet("i")
# i.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
m.addSet("i")
# m.data["i"].setRecords(["seattle", "san-diego"])
In [1]: i.records
Out[1]:
0 seattle
1 san-diego
```

Example #2 - Create a 1D set from a tuple

```python
import gamstransfer as gt
m = gt.Container()
j = gt.Set(m, "j", records=("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
j1 = gt.Set(m, "j")
# j.setRecords(("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
m.addSet("j", records=("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
j1 = m.addSet("j")
# j.setRecords(("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
m.addSet("j")
# m.data["j"].setRecords(("seattle", "san-diego"))
In [1]: j.records
Out[1]:
0 seattle
1 san-diego
```

Example #3 - Create a 2D set from a list of tuples

```python
import gamstransfer as gt
m = gt.Container()
k = gt.Set(m, "k", ["*", "*"], records=[["seattle", "san-diego"]])
# NOTE: the above syntax is equivalent to -
k1 = gt.Set(m, "k")
# k.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
m.addSet("k", ["*", "*"], records=[["seattle", "san-diego"]])
# NOTE: the above syntax is also equivalent to -
k1 = m.addSet("k")
# k.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
k1 = m.addSet("k")
# m.data["k"].setRecords(["seattle", "san-diego"])
In [1]: k.records
Out[1]:
0 seattle san-diego
```
Example #4 - Create a 1D set from a DataFrame slice + .unique()

```python
import gamstransfer as gt
m = gt.Container()
# note that the raw data is convenient to hold in a DataFrame
dist = pd.DataFrame([('seattle', 'new-york', 2.5),
                     ('seattle', 'chicago', 1.7),
                     ('seattle', 'topeka', 1.8),
                     ('san-diego', 'new-york', 2.5),
                     ('san-diego', 'chicago', 1.8),
                     ('san-diego', 'topeka', 1.4),
                 ],
                 columns=['from', 'to', 'thousand_miles'],)
l = gt.Set(m, "l", records=dist['from'].unique())
# NOTE: the above syntax is equivalent to -
# l = gt.Set(m, "l")
# l.setRecords(dist['from'].unique())
# NOTE: the above syntax is also equivalent to -
# m.addSet('l', records=dist['from'].unique())
# NOTE: the above syntax is also equivalent to -
# l = m.addSet('l')
# NOTE: the above syntax is also equivalent to -
# m.data['l'].setRecords(dist['from'].unique())
In [1]: l.records
Out[1]:
0    seattle
1    san-diego
```

Note
The .unique() method preserves the order of appearance, unlike set().

Set element text is very handy when labeling specific set elements within a set. A user can add a set element text directly with a set element. Note that it is not required to label all set elements, as can be seen in the following example.

Example #5 - Add set element text

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i",
           records=[
                       ('seattle', "home of sub pop records"),
                       ('san-diego'),
                       ('washington_dc', "former gams hq"),
                   ],
           )
# NOTE: the above syntax is equivalent to -
# i = gt.Set(m, "i")
# i.recs = [
#     ('seattle', "home of sub pop records"),
#     ('san-diego'),
#     ('washington_dc', "former gams hq"),
# ]
# i
# i.setRecords(i.recs)
# NOTE: the above syntax is also equivalent to -
# m.addSet('i', records=i.recs)
# NOTE: the above syntax is also equivalent to -
# i = m.addSet('i')
# i.setRecords(i.recs)
# NOTE: the above syntax is also equivalent to -
# m.data['i'].setRecords(i.recs)
In [1]: i.records
Out[1]:
0    seattle  home of sub pop records
1    san-diego
2    washington_dc  former gams hq
```
7.5.2.5.5 Directly Set Records  The primary advantage of the setRecords method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the Container a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Directly set records (1D set)

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", description="supply")
# create a standard format dataframe
df_i = pd.DataFrame(
    data=[("seattle", ""), ("san-diego", "")], columns=["uni_0", "element_text"]
)
# need to create categorical column type, referencing elements already in df_i
df_i["uni_0"] = df_i["uni_0"]).astype(
    pd.CategoricalDtype(categories=df_i["uni_0"]).unique(), ordered=True)
# set the records directly
i.records = df_i
In [1]: i.isValid()
Out[1]: True
```

Stepping through this example we take the following steps:

1. Create an empty Container
2. Create a GAMS set i in the Container, but do not set the records
3. Create a Pandas DataFrame (manually, in this example) taking care to follow the standard format
4. The DataFrame has the right shape and column labels so we can proceed to set the records.
5. We need to cast the uni_0 column as a categorical data type, so we create a custom ordered category type using pandas.CategoricalDtype
6. Finally, we set the records directly by passing a reference to df_i into the symbol records attribute.

The setter function of .records checks that a DataFrame is being set, but does not check validity.

Thus, as a final step we call the .isValid() method to verify that the symbol is valid.

Attention

Users can debug their DataFrames by running <symbol_name>.isValid(verbosetrue) to get feedback about their data.

Example #2 - Directly set records (1D subset)

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", i, description="supply")
# create a standard format dataframe
df_j = pd.DataFrame(data=[("seattle", "")], columns=["i_0", "element_text"])
# create the categorical column type
df_j["i_0"] = df_j["i_0"]).astype(i.records["uni_0"]).dtype
# set the records
j.records = df_j
In [1]: j.isValid()
Out[1]: True
```

This example is more subtle in that we want to create a set j that is a subset of i. We create the set i using the setRecords method but then set the records directly for j. There are two important details to note: 1) the column labels in df_j now reflect the standard format for a symbol with a domain set (as opposed to the universe) and 2) we create the categorical dtype by referencing the parent set (i) for the categories (instead of referencing itself).
7.5.2.6  Create a Parameter

There are two different ways to create a GAMS parameter and add it to a Container.

1. Use Parameter constructor
2. Use the Container method addParameter (which internally calls the Parameter constructor)

7.5.2.6.1  Parameter Constructor

Parameter constructor

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>List of domains given either as string ('*' for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar parameter</td>
<td>No</td>
<td>[]</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>bool</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>False</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;*&quot;] * n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain</td>
<td>list of domains given either as string ('*' for universe set) or as reference to the Set/Alias object</td>
<td>list</td>
<td>-</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>is_scalar</td>
<td>True if the len(self.domain) = 0</td>
<td>bool</td>
<td>-</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Type</td>
<td>Special Setter Behavior</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------</td>
<td>-----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., return len(self.records) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>ref_container</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
<td>-</td>
</tr>
<tr>
<td>shape</td>
<td>a tuple describing the array dimensions if records were converted with .toDense()</td>
<td>tuple</td>
<td>-</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>

### 7.5.2.6.2 Parameter Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues.EPS in value column</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues.NA in value column</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues.NEGINF in value column</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues.POSINF in value column</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues.UNDEF in value column</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>returns the count of how many records contain at least one domain violation</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records from the symbol that contain a domain violation</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are duplicate records in the symbol, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from the symbol – keep argument can take values of &quot;first&quot; (keeps the first instance of a duplicate record), &quot;last&quot; (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>findDuplicateRecords</td>
<td>get the index of records that contain any duplicate domains – keep argument can take values of &quot;first&quot; (finds all duplicates while keeping the first instance as unique), &quot;last&quot; (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)</td>
<td>keep=&quot;first&quot;</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>findEps</td>
<td>find index positions of SpecialValues.EPS in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of SpecialValues.NA in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of SpecialValues.NEGINF in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of SpecialValues.POSINF in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of SpecialValues.Undef in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value in value column</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value in value column</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value in value column</td>
<td>-</td>
<td>float or None</td>
</tr>
</tbody>
</table>
### 7.5.2.6.3 Parameter Methods

#### 7.5.2.6.4 Adding Parameter Records

Three possibilities exist to assign symbol records to a parameter (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy
3. Setting the property `records` directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the parameter constructor. This is an optional keyword argument and internally the parameter constructor will simply call the `setRecords` method. The symbol method `setRecords` is a convenience method that transforms the given data into an approved Pandas DataFrame format (see GAMS Transfer Standard Data Formats). Many native python data types can be easily transformed into DataFrames, so the `setRecords` method for `Set` objects will accept a number of different types for input. The `setRecords` method is called internally on any data structure that is passed through the `records` argument. We show a few examples of ways to create differently structured parameters:
Example #1 - Create a GAMS scalar

```python
import gamstransfer as gt
m = gt.Container()
pi = gt.Parameter(m, "pi", records=3.14159)
# NOTE: the above syntax is equivalent to -
# pi = gt.Parameter(m, "pi")
# pi.setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m.addParameter("pi")
# m.setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m.data["pi"].setRecords(3.14159)
In [14]: pi.records
Out[14]:
value
0 3.14159
```

Note

GAMS Transfer will still convert scalar values to a standard format (i.e., a Pandas DataFrame with a single row and column).

Example #2 - Create a 1D parameter (defined over *) from a list of tuples

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Parameter(m, "i", ["*"], records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is equivalent to -
# i = gt.Parameter(m, "i")
# i.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addParameter("i")
# m.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# i = m.addParameter("i")
# i.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.data["i"].setRecords([("i" + str(i), i) for i in range(5)])
In [1]: i.records
Out[1]:
uni
0 value
0 i0 0.0
1 i1 1.0
2 i2 2.0
3 i3 3.0
4 i4 4.0
```

Example #3 - Create a 1D parameter (defined over a set) from a list of tuples

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
a = gt.Parameter(m, "a", i, records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is equivalent to -
# i = gt.Set(m, "i")
# i.setRecords(["i" + str(i) for i in range(5)])
# a = gt.Parameter(m, "a", i)
# a.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i")
# m.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# a = m.addParameter("a")
# a.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.data["a"].setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i")
```
Example #4 - Create a 2D parameter (defined over a set) from a DataFrame slice

```python
import gamstransfer as gt
import pandas as pd

dist = pd.DataFrame(
    [('seattle', 'new-york', 2.5),
     ('seattle', 'chicago', 1.7),
     ('seattle', 'topeka', 1.8),
     ('san-diego', 'new-york', 2.5),
     ('san-diego', 'chicago', 1.8),
     ('san-diego', 'topeka', 1.4),
    ],
    columns=['from', 'to', 'thousand miles'],
)

m = gt.Container()
i = gt.Set(m, "i", ['*'], records=dist['from'].unique())
j = gt.Set(m, "j", ['*'], records=dist['to'].unique())
a = gt.Parameter(m, "a", [i, j], records=dist.loc[[0, 3], :])

In [1]: a.records
Out[1]:
0 j
1 value
0 seattle new-york 2.5
3 san-diego new-york 2.5
```

Note
---
The original indexing is preserved when a user slices rows out of a reference dataframe.

Example #5 - Create a 2D parameter (defined over a set) from a matrix

```python
import gamstransfer as gt
import pandas as pd

dist = pd.DataFrame(
    [('seattle', 'new-york', 2.5),
     ('seattle', 'chicago', 1.7),
     ('seattle', 'topeka', 1.8),
     ('san-diego', 'new-york', 2.5),
     ('san-diego', 'chicago', 1.8),
     ('san-diego', 'topeka', 1.4),
    ],
    columns=['from', 'to', 'thousand miles'],
)

m = gt.Container()
i = gt.Set(m, "i", ['*'], records=dist['from'].unique())
j = gt.Set(m, "j", ['*'], records=dist['to'].unique())
a = gt.Parameter(m, "a", [i, j], records=dist)  

In [1]: a.toDense()
Out[1]:
array([[2.5, 1.7, 1.8],
       [2.5, 1.8, 1.4]])
```

# use a.toDense() to create a new (and identical) parameter a2
a2 = gt.Parameter(m, "a2", [i, j], records=a.toDense())
# check that a.records is identical to a2.records
In [1]: a.records.equals(a2.records)
Out[1]: True
Example #6 - Create a 2D parameter from an array using setRecords

```python
import gamstransfer as gt
import numpy as np
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", records=["j" + str(j) for j in range(5)])
# create the parameter with linked domains (these will control the .shape of the symbol)
a = gt.Parameter(m, "a", [i, j])
# here we use the .shape property to easily generate a dense random array in numpy
a.setRecords(np.random.uniform(low=1, high=10, size=a.shape))
```

```
In [1]: a.toDense()
Out[1]:
array([[3.66944951, 5.17395381, 1.99129484, 3.28315433, 1.44793791],
       [1.06953243, 6.56331121, 5.26162554, 5.98098795, 8.30006       ],
       [3.77213221, 5.82144901, 9.30035479, 9.12534285, 8.51970747],
       [8.47965504, 7.84426304, 5.24424710, 6.96666622, 6.55194415],
       [5.6282779, 4.92509183, 8.94579609, 2.77249344, 9.99576081]])
```

7.5.2.6.5 Directly Set Records  
As with sets, the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Correctly set records (directly)

```python
import gamstransfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(data=[[h + str(h), m + str(m), s + str(s)] for h in range(8760) for m in range(60) for s in range(60)], columns=["h","m","s"],
                 index=[h in range(8760) for h in range(8760)] for m in range(60) for s in range(60)],
                 columns=["h","m","s"],
                 )
df["value"] = np.random.uniform(0, 100, len(df))

hrs = gt.Set(m, "h", records=df["h"].unique())
mns = gt.Set(m, "m", records=df["m"].unique())
secs = gt.Set(m, "s", records=df["s"].unique())
df["h"] = df["h"].astype(hrs.records["uni0"].dtype)
df["m"] = df["m"].astype(mns.records["uni0"].dtype)
df["s"] = df["s"].astype(secs.records["uni0"].dtype)
a = gt.Parameter(m, "a", [hrs, mins, secs])
# set records
a.records = df
In [1]: a.isValid()
Out[1]: True
```

In this example we create a large parameter (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to a parameter with `a.records`. GAMS Transfer requires that all domain columns must be a categorical data type, furthermore, this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`.

Users will need to use the `.isValid(verbos=True)` method to debug any structural issues. As an example we incorrectly generate categorical data types by passing the DataFrame constructor the generic `dtype="category"` argument. This creates categorical column types but they are not ordered and they do not reference the underlying domain set. These errors result in `a` being invalid.
Example #2 - Incorrectly set records (directly)

```python
import gamstransfer as gt
import pandas as pd
import numpy as np

df = pd.DataFrame(
    data=[
        ("h" + str(h), "m" + str(m), "s" + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=["h_0", "m_1", "s_2"],
    dtype="category"
)
df["value"] = np.random.uniform(0, 100, len(df))
a = gt.Container()
hrs = gt.Set(a, "h", records=df["h_0"].unique())
mins = gt.Set(a, "m", records=df["m_1"].unique())
secs = gt.Set(a, "s", records=df["s_2"].unique())
a = gt.Parameter(a, "a", [hrs, mins, secs])
# set the records directly
a.records = df
In [1]: a.isValid()
Out[1]: False
In [2]: a.isValid(verbose=True)
Out[2]: Exception: Domain information in column 'h_0' for 'records' must be an ORDERED categorical type (i.e., <symbol_object>.records[h_0].dtype.ordered = True)
```

7.5.2.7 Create a Variable

There are two different ways to create a GAMS variable and add it to a `Container`.

1. Use `Variable` constructor
2. Use the `Container` method `addVariable` (which internally calls the `Variable` constructor)

7.5.2.7.1 Variable Constructor

Variable constructor

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the <code>Container</code> object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str</td>
<td>Type of variable being created [binary, integer, positive, negative, free, sos1, sos2, semicont, semiint]</td>
<td>No</td>
<td>free</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>List of domains given either as string (= for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar variable</td>
<td>No</td>
<td>[]</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>domain_forwading</td>
<td>bool</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>False</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>
### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;\ast]\ \ast n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain</td>
<td>list of domains given either as string (\ast for universe set) or as reference to the Set/Alias object</td>
<td>list</td>
<td>-</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns (\text{len(self.records)}) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>ref_container</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
<td>-</td>
</tr>
<tr>
<td>shape</td>
<td>a tuple describing the array dimensions if records were converted with .toDense()</td>
<td>tuple</td>
<td>-</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str type of variable</td>
<td>str</td>
<td>-</td>
</tr>
</tbody>
</table>

### 7.5.2.7.2 Variable Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues.EPS across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues.NA across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues.NEGINF across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues.POSINF across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues.UNDEF across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>returns the count of how many records contain at least one domain violation</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records from the symbol that contain a domain violation</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are duplicate records in the symbol, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from the symbol – keep argument can take values of &quot;first&quot; (keeps the first instance of a duplicate record), &quot;last&quot; (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>findDuplicateRecords</td>
<td>get the index of records that contain any duplicate domains – keep argument can take values of &quot;first&quot; (finds all duplicates while keeping the first instance as unique), &quot;last&quot; (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)</td>
<td>keep=&quot;first&quot;</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>findEps</td>
<td>find index positions of SpecialValues.EPS in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of SpecialValues.NA in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of SpecialValues.NEGINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of SpecialValues.POSINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of SpecialValues.Undef in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=True, recheck a symbol if force=True</td>
<td>verbose=False, force=True</td>
<td>bool</td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard pandas.DataFrame records</td>
<td>records (many types)</td>
<td>None</td>
</tr>
<tr>
<td>toDense</td>
<td>convert column to a dense numpy.array format</td>
<td>column=&quot;level&quot; (str)</td>
<td>numpy.array or None</td>
</tr>
<tr>
<td>toSparseCoo</td>
<td>convert column to a sparse COOrdinate numpy.array format</td>
<td>column=&quot;level&quot; (str)</td>
<td>sparse matrix format or None</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the domain entry of records with a maximum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the domain entry of records with a maximum absolute value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the domain entry of records with a minimum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
</tbody>
</table>
7.5.2.7.3 Variable Methods

7.5.2.7.4 Adding Variable Records

Three possibilities exist to assign symbol records to a variable (roughly ordered in complexity):

1. Setting the argument records in the set constructor/container method (internally calls setRecords) - creates a data copy
2. Using the symbol method setRecords - creates a data copy
3. Setting the property records directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the variable constructor. This is an optional keyword argument and internally the variable constructor will simply call the setRecords method. In contrast to the setRecords methods in either the Set or Parameter classes the setRecords method for variables will only accept Pandas DataFrames and specially structured dict for creating records from matrices. This restriction is out of necessity because to properly set a record for a Variable the user must pass data for the level, marginal, lower, upper and scale attributes. That said, any missing attributes will be filled in with the GAMS default record values (see: Variable Types), default scale value is always 1, and the default level and marginal values are 0 for all variable types).

We show a few examples of ways to create differently structured variables:

Example #1 - Create a GAMS scalar variable

```python
import gamstransfer as gt
m = gt.Container()
pi = gt.Variable(m, "pi", records=pd.DataFrame(data=[3.14159], columns=["level"])
# NOTE: the above syntax is equivalent to -
# pi = gt.Variable(m, "pi", "free")
# pi.setRecords(pd.DataFrame(data=[3.14159], columns=["level"])
# NOTE: above syntax also equivalent to -
# m.addVariable("pi", "free", records=pd.DataFrame(data=[3.14159], columns=["level"]))
In [1]: pi.records
Out[1]:
uni
0 level marginal lower upper scale
0 3.14159 0.0 -inf inf 1.0
```

Example #2 - Create a 1D variable (defined over *) from a list of tuples

In this example we only set the marginal values.

```python
import gamstransfer as gt
m = gt.Container()
v = gt.Variable(
    m,
    "v",
    "free",
    domain=["*"],
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(6)], columns=["domain", "marginal"]
    ),
)
In [1]: v.records
Out[1]:
uni
0 level marginal lower upper scale
0 0.0 0.0 -inf inf 1.0
1 11.0 1.0 -inf inf 1.0
2 12.0 2.0 -inf inf 1.0
3 13.0 3.0 -inf inf 1.0
4 14.0 4.0 -inf inf 1.0
```
Example #3 - Create a 1D variable (defined over a set) from a list of tuples

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"]
records=["i" + str(i) for i in range(5)])
v = gt.Variable(m, "v", "free",
domain=i,
records=pd.DataFrame(
data=["i" + str(i) for i in range(5)], columns=["domain", "marginal"]
), )
In [1]: v.records
Out[1]:
   i  level  marginal lower upper scale
0  0    0.0      0.0  -inf  inf  1.0
1  1    0.0      1.0  -inf  inf  1.0
2  2    0.0      2.0  -inf  inf  1.0
3  3    0.0      3.0  -inf  inf  1.0
4  4    0.0      4.0  -inf  inf  1.0
```

Example #4 - Create a 2D positive variable, specifying no numerical data

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
v = gt.Variable(m, "v", "positive",
records=pd.DataFrame(["seattle", "san-diego"], "chicago", "madison")
),
In [1]: v.records
Out[1]:
   uni  level  marginal lower upper scale
0  san-diego    0.0      0.0  0.0  inf  1.0
1  madison      0.0      0.0  0.0  inf  1.0
```

Example #5 - Create a 2D variable (defined over a set) from a matrix

```python
import gamstransfer as gt
import pandas as pd
import numpy as np
m = gt.Container()
i = gt.Set(m, "i", ["*"]
records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", ["*"]
records=["j" + str(i) for i in range(5)])
a = gt.Parameter(m, "a", 
i, j,
records=["i" + str(i) for i in range(5)], j + j for j in range(5)],
) = a.toDense()
# create a free variable and set the level and marginal attributes from matrices
v = gt.Variable(m, "v", domain=[i, j], records={"level": a.toDense(), "marginal": a.toDense()} )
# if not specified, the toDense() method will convert the level values to a matrix
In [1]: v.toDense()
Out[1]:
   0  1  2  3  4
0  0  1  2  3  4
1  1  2  3  4  5
2  2  3  4  5  6
3  3  4  5  6  7
4  4  5  6  7  8
```

7.5.2.7.5 Directly Set Records As with sets, the primary advantage of the setRecords method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the Container a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.
Example #1 - Correctly set records (directly)

```python
import gamstransfer as gt
import pandas as pd
import numpy as np

df = pd.DataFrame(
    data=[
        ('h' + str(h), 'm' + str(m), 's' + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=['h_0', 'm_1', 's_2'],
)
# it is necessary to specify all variable attributes if setting records directly
# NOTE: all numeric data must be type float
df['level'] = np.random.uniform(0, 100, len(df))
df['marginal'] = 0.0
df['lower'] = gt.SpecialValues.NEGINF
df['upper'] = gt.SpecialValues.POSINF
df['scale'] = 1.0

m = gt.Container()
hrs = gt.Set(m, 'h', records=df['h_0'].unique())
mins = gt.Set(m, 'm', records=df['m_1'].unique())
secs = gt.Set(m, 's', records=df['s_2'].unique())
df['h_0'] = df['h_0'].astype(hrs.records['universe'].dtype)
df['m_1'] = df['m_1'].astype(mins.records['universe'].dtype)
df['s_2'] = df['s_2'].astype(secs.records['universe'].dtype)
a = gt.Variable(m, 'a', domain=[hrs, mins, secs])
# set records
a.records = df
# in [1]: a.isValid()
Out[1]: True
```

Attention

All numeric data in the records will need to be type float in order to maintain a valid symbol.

In this example we create a large variable (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to a variable with `a.records`. GAMS Transfer requires that all domain columns must be a categorical data type, furthermore this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`. As with Set and Parameters, users can use the `.isValid(verbos=True)` method to debug any structural issues.

7.5.2.8 Create an Equation

There are two different ways to create a GAMS equation and add it to a `Container`.

1. Use `Equation` constructor
2. Use the `Container` method `addEquation` (which internally calls the `Equation` constructor)

7.5.2.8.1 Equation Constructor

Equation constructor
### 7.5 Python API

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str</td>
<td>Type of equation being created [eq (or E/e), geq (or G/g), leq (or L/l), nonbinding (or N/n), external (or X/x)]</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>List of domains given either as string (* for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar equation</td>
<td>No</td>
<td>[]</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>bool</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>False</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;*&quot;*] * n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain</td>
<td>list of domains given either as string (* for universe set) or as reference to the Set/Alias object</td>
<td>list</td>
<td>-</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns len(self.records) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>ref_container</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
<td>-</td>
</tr>
<tr>
<td>shape</td>
<td>a tuple describing the array dimensions if records were converted with .toDense()</td>
<td>tuple</td>
<td>-</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Type</td>
<td>Special Setter Behavior</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------</td>
<td>--------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str type of variable</td>
<td>str</td>
<td>-</td>
</tr>
</tbody>
</table>

### 7.5.2.8.2 Equation Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of (\text{SpecialValues.EPS}) across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of (\text{SpecialValues.NA}) across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of (\text{SpecialValues.NEGINF}) across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of (\text{SpecialValues.POSINF}) across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of (\text{SpecialValues.UNDEF}) across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>int or None</td>
</tr>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>returns the count of how many records contain at least one domain violation</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records from the symbol that contain a domain violation</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are duplicate records in the symbol, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from the symbol – keep argument can take values of &quot;first&quot; (keeps the first instance of a duplicate record), &quot;last&quot; (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>findDuplicateRecords</td>
<td>get the index of records that contain any duplicate domains – keep argument can take values of &quot;first&quot; (finds all duplicates while keeping the first instance as unique), &quot;last&quot; (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)</td>
<td>keep=&quot;first&quot;</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>findEps</td>
<td>find index positions of SpecialValues.EPS in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of SpecialValues.NA in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of SpecialValues.NEGINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of SpecialValues.POSINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of SpecialValues.Undef in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns=&quot;level&quot; (str,list)</td>
<td>float or None</td>
</tr>
</tbody>
</table>
### 7.5.2.8.3 Equation Methods

#### 7.5.2.8.4 Adding Equation Records

Adding equation records mimics that of variables – three possibilities exist to assign symbol records to an equation (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy
3. Setting the property `records` directly - does not create a data copy

Setting equation records require the user to be explicit with the type of equation that is being created; in contrast to setting variable records (where the default variable is considered to be `free`).

If the data is in a convenient format, a user may want to pass the records directly within the equation constructor. This is an optional keyword argument and internally the equation constructor will simply call the `setRecords` method. In contrast to the `setRecords` methods in in either the Set or Parameter classes the `setRecords` method for variables will only accept Pandas DataFrames and specially structured `dict` for creating records from matrices. This restriction is out of necessity because to properly set a record for an Equation the user must pass data for the `level, marginal, lower, upper` and `scale` attributes. That said, any missing attributes will be filled in with the GAMS default record values (`level = 0.0, marginal = 0.0, lower = -inf, upper = inf, scale = 1.0`). We show a few examples of ways to create differently structured variables:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isValid</code></td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=True, recheck a symbol if force=True</td>
<td>verbose=False, force=True</td>
<td>bool</td>
</tr>
<tr>
<td><code>setRecords</code></td>
<td>main convenience method to set standard pandas.DataFrame records</td>
<td>records (many types)</td>
<td>None</td>
</tr>
<tr>
<td><code>toDense</code></td>
<td>convert column to a dense numpy.array format</td>
<td>column=&quot;level&quot; (str)</td>
<td>numpy.array or None</td>
</tr>
<tr>
<td><code>toSparseCoo</code></td>
<td>convert column to a sparse COOrdinate numpy.array format</td>
<td>column=&quot;level&quot; (str)</td>
<td>sparse matrix format or None</td>
</tr>
<tr>
<td><code>whereMax</code></td>
<td>find the domain entry of records with a maximum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td><code>whereMaxAbs</code></td>
<td>find the domain entry of records with a maximum absolute value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td><code>whereMin</code></td>
<td>find the domain entry of records with a minimum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
</tbody>
</table>
Example #1 - Create a GAMS scalar equation

```python
import gamstransfer as gt
m = gt.Container()
# here we create an equality (=E=) equation
z = gt.Equation(m, "z", "eq", records=pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is equivalent to -
# pi = gt.Equation(m, "pi", "eq")
# pi.setRecords(pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is also equivalent to -
# m.addEquation("pi", "eq", records=pd.DataFrame(data=[3.14159], columns=["level"]))
In [1]: pi.records
Out[1]:
          level  marginal    lower    upper    scale
         -------  --------  --------  --------  -----
0  3.14159   0.00000  -inf       inf     1.0
```

Example #2 - Create a 1D Equation (defined over *) from a list of tuples

```python
In this example we only set the marginal values.
import gamstransfer as gt
m = gt.Container()
# here we define a greater than or equal (=G=) equation
i = gt.Equation(
    m,
    "i",
    "geq",
    domain=["*"],
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)],
        columns=["domain", "marginal"]
    ),
)
In [1]: i.type
Out[1]: 'geq'
In [2]: i.records
Out[2]:
       uni
0 level  marginal    lower    upper    scale
0  0.00000  0.00000  -inf       inf     1.0
1  1.00000  1.00000  -inf       inf     1.0
2  2.00000  2.00000  -inf       inf     1.0
3  3.00000  3.00000  -inf       inf     1.0
4  4.00000  4.00000  -inf       inf     1.0
```

Example #3 - Create a 1D Equation (defined over a set) from a list of tuples

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
# here we define a less than or equal (=L=) equation
e = gt.Equation(
    m,
    "e",
    "leq",
    domain=i,
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)],
        columns=["domain", "marginal"]
    ),
)
In [1]: i.type
Out[1]: 'leq'
In [5]: e.records
Out[5]:
       uni
0 level  marginal    lower    upper    scale
0  0.00000  0.00000  -inf       inf     1.0
1  1.00000  1.00000  -inf       inf     1.0
2  2.00000  2.00000  -inf       inf     1.0
3  3.00000  3.00000  -inf       inf     1.0
4  4.00000  4.00000  -inf       inf     1.0
```
Example #4 - Create a 2D equation, specifying no numerical data

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
e = gt.Equation(m,
                "e",
                "eq",
                "[*] * ["]",
                records=pd.DataFrame([("seattle", "san-diego"), ("chicago", "madison")]),
)
In [1]: e.records
Out[1]:
    uni   0  uni   1  level  marginal  lower  upper  scale
 0 seattle san-diego  0.0  0.0  -inf   inf   1.0
 1 chicago  madison  0.0  0.0  -inf   inf   1.0
```

Example #5 - Create a 2D equation (defined over a set) from a matrix

```python
import gamstransfer as gt
import pandas as pd
import numpy as np
m = gt.Container()
i = gt.Set(m, "i", ["*"]
                records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", ["*"]
                records=["j" + str(i) for i in range(5)])
a = gt.Parameter(m,
                "a",
                [i, j],
                records=[("i" + str(i), "j" + str(j), i + j) for i in range(5) for j in range(5)],
)
# create a nonbinding (=N=) equation and set the level and marginal attributes from matrices
e = gt.Equation(m,
                "e",
                "nonbinding",
                domain=[i, j],
                records=["level": a.toDense(), "marginal": a.toDense()}
)
In [1]: e.records
Out[1]:
    i  j 0  j 1  level  marginal  lower  upper  scale
 0  i 0  j 1  1.0  1.0  -inf   inf   1.0
 1  i 0  j 2  2.0  2.0  -inf   inf   1.0
 2  i 1  j 0  3.0  3.0  -inf   inf   1.0
 3  i 1  j 1  4.0  4.0  -inf   inf   1.0
 4  i 1  j 2  5.0  5.0  -inf   inf   1.0
 5  i 1  j 3  6.0  6.0  -inf   inf   1.0
 6  i 2  j 0  7.0  7.0  -inf   inf   1.0
 7  i 2  j 1  8.0  8.0  -inf   inf   1.0
 8  i 2  j 2  9.0  9.0  -inf   inf   1.0
 9  i 2  j 3 10.0 10.0  -inf   inf   1.0
10  i 3  j 0 11.0 11.0  -inf   inf   1.0
11  i 3  j 1 12.0 12.0  -inf   inf   1.0
12  i 3  j 2 13.0 13.0  -inf   inf   1.0
13  i 3  j 3 14.0 14.0  -inf   inf   1.0
14  i 4  j 0 15.0 15.0  -inf   inf   1.0
15  i 4  j 1 16.0 16.0  -inf   inf   1.0
16  i 4  j 2 17.0 17.0  -inf   inf   1.0
17  i 4  j 3 18.0 18.0  -inf   inf   1.0
18  i 4  j 4 19.0 19.0  -inf   inf   1.0
19  i 5  j 0 20.0 20.0  -inf   inf   1.0
20  i 5  j 1 21.0 21.0  -inf   inf   1.0
21  i 5  j 2 22.0 22.0  -inf   inf   1.0
22  i 5  j 3 23.0 23.0  -inf   inf   1.0
23  i 5  j 4 24.0 24.0  -inf   inf   1.0
# if not specified, the toDense() method will convert the level values to a matrix
In [2]: e.toDense()
Out[2]:
array([[0., 1., 2., 3., 4.],
       [1., 2., 3., 4., 5.],
       [2., 3., 4., 5., 6.],
       [3., 4., 5., 6., 7.],
       [4., 5., 6., 7., 8.]])
```

7.5.2.8.5 Directly Set Records As with set, parameters and variables, the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these
computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Correctly set records (directly)

```python
import gamstransfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(
    data=[
        ('h' + str(h), 'm' + str(m), 's' + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=['h0', 'm1', 's2']
)
# it is necessary to specify all variable attributes if setting records directly
# NOTE: all numeric data must be type float
df['level'] = np.random.uniform(0, 100, len(df))
df['marginal'] = 0.0
df['lower'] = gt.SpecialValues.NEGINF
df['upper'] = gt.SpecialValues.POSINF
df['scale'] = 1.0
m = gt.Container()
hrs = gt.Set(m, 'h', records=df['h0'].unique())
mins = gt.Set(m, 'm', records=df['m1'].unique())
secs = gt.Set(m, 's', records=df['s2'].unique())
df['h0'] = df['h0'].astype(hrs.records['uni0'].dtype)
df['m1'] = df['m1'].astype(mins.records['uni0'].dtype)
df['s2'] = df['s2'].astype(secs.records['uni0'].dtype)
a = gt.Equation(m, 'a', 'eq', domain=[hrs, mins, secs])
# set records
a.records = df
```

In [1]: e.isValid()
Out[1]: True

Attention

All numeric data in the records will need to be type float in order to maintain a valid symbol.

In this example we create a large equation (31,536,000 records and 8880 unique domain elements) and assign it to a variable with `a.records`. GAMS Transfer requires that all domain columns must be a categorical data type, furthermore this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`. As with Set and Parameters, users can use the `.isValid(verbos=True)` method to debug any structural issues.

7.5.2.9 Create an Alias

There are two different ways to create a GAMS equation and add it to a `Container`.

1. Use `Alias` constructor
2. Use the `Container` method `addAlias` (which internally calls the `Alias` constructor)

7.5.2.9.1 Alias Constructor

Alias constructor
### Application Programming Interfaces

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>alias_with</td>
<td>Set object</td>
<td>set object from which to create an alias</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example - Creating an alias from a set**

GAMS Transfer only stores the reference to the parent set as part of the alias structure – most properties that are called from an alias object simply point to the properties of the parent set (with the exception of `ref_container`, `name`, and `alias_with`). It is possible to create an alias from another alias object. In this case a recursive search will be performed to find the root parent set – this is the set that will ultimately be stored as the `alias_with` property. We can see this behavior in the following example:

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records="i" + str(i) for i in range(5))
ip = gt.Alias(m, "ip", i)
ipp = gt.Alias(m, "ipp", ip)

In [1]: ip.alias
Out[1]: 'i'
In [2]: ipp.alias
Out[2]: 'i'
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias_with</td>
<td>aliased object</td>
<td>Set</td>
<td>-</td>
</tr>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create <code>[*s] * n</code> domains in symbol</td>
</tr>
<tr>
<td>domain_forwaring</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>is_singleton</td>
<td>if symbol is a singleton set</td>
<td>bool</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns <code>len(self.records)</code> if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to <code>domain_forwaring</code> state</td>
</tr>
<tr>
<td>ref_container</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
<td>-</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>
### 7.5 Python API

#### 7.5.2.9.2 Alias Properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasDomainViolations</td>
<td>returns True if there are domain violations in the records of the parent set, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDomainViolations</td>
<td>returns the count of how many records in the parent set contain at least one domain violation</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDomainViolations</td>
<td>drop records from the parent set that contain a domain violation</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records from the parent set that contain any domain violations</td>
<td>-</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>hasDuplicateRecords</td>
<td>returns True if there are duplicates in the records of the parent set, returns False if not.</td>
<td>-</td>
<td>bool</td>
</tr>
<tr>
<td>countDuplicateRecords</td>
<td>returns the count of how many duplicate records exist in the parent set</td>
<td>-</td>
<td>int</td>
</tr>
<tr>
<td>dropDuplicateRecords</td>
<td>drop records with duplicate domains from the parent set – keep argument can take values of &quot;first&quot; (keeps the first instance of a duplicate record), &quot;last&quot; (keeps the last instance of a record), or False (drops all duplicates including the first and last)</td>
<td>keep=&quot;first&quot;</td>
<td>None</td>
</tr>
<tr>
<td>findDuplicateRecords</td>
<td>get the index of records from the parent set that contain any duplicate domains – keep argument can take values of &quot;first&quot; (finds all duplicates while keeping the first instance as unique), &quot;last&quot; (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)</td>
<td>keep=&quot;first&quot;</td>
<td>pandas.Index</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=True, re-check a symbol if force=True</td>
<td>verbose=True, force=True</td>
<td>bool</td>
</tr>
</tbody>
</table>
### 7.5.2.9.3 Alias Methods

#### 7.5.2.9.4 Adding Alias Records

The linked structure of Aliases offers some unique opportunities to access some of the setter functionality of the parent set. Specifically, GAMS Transfer allows the user to change the domain, description, dimension, and records of the underlying parent set as a shorthand notation. We can see this behavior if we look at a modified Example #1 from Adding Set Records.

Example - Creating set records through an alias link

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i")
ip = gt.Alias(m, "ip", i)
ip.description = "adding new descriptive set text"
ip.domain = ['*', '*']
ip.setRecords([("i" + str(i), "j" + str(j)) for i in range(3) for j in range(3)])
In [1]: i.description
Out[1]: 'adding new descriptive set text'
In [2]: i.domain
Out[2]: ['*', '*']
In [3]: i.records
Out[3]:
   0  10  j0
   1  10  j1
   2  10  j2
   3  11  j0
   4  11  j1
   5  11  j2
   6  12  j0
   7  12  j1
   8  12  j2
```

Note

An alias .isValid()=True when the underlying parent set is also valid – if the parent set is removed from the Container the alias will no longer be valid.

### 7.5.2.10 Validating Data

GAMS Transfer requires that the records for all symbols exist in a standard format (GAMS Transfer Standard Data Formats) in order for them to be understood and written successfully. It is certainly possible that the data could end up in a state that is inconsistent with the standard format (especially if setting symbol attributes directly). GAMS Transfer includes the .isValid() method in order to determine if a symbol is valid and ready for writing; this method returns a bool. For example, we create two valid sets and then check them with .isValid() to be sure.

Note

It is possible to run .isValid() on both the Container as well as the symbol object – .isValid() will also return a bool if there are any invalid symbols in the Container object.
Example (valid data)

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=\"seattle", \"san-diego", \"washington dc\")
j = gt.Set(m, "j", i, records=\"san-diego", \"washington dc\")
In [1]: i.isValid()
Out[1]: True
In [2]: j.isValid()
Out[2]: True
In [3]: m.isValid()
Out[3]: True
```

Now we create some data that is invalid due to domain violations in the set j.

Example (intentionally create domain violations)

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=\"seattle", \"san-diego", \"washington dc\")
j = gt.Set(m, "j", i, records=\"grayslake", \"washington dc\")
In [1]: i.isValid()
Out[1]: True
In [2]: j.isValid()
Out[2]: False
In [3]: m.isValid()
Out[3]: False
```

In this example, we know that the validity of the data is compromised by the domain violations, but there could be other subtle discrepancies that must be remedied before writing data. The user can get more detailed error reporting if the `verbose` argument is set to `True`. For example:

```python
In [1]: j.isValid(verbose=True)
```

Exception: Symbol 'records' contain domain violations; ensure that all domain elements have been mapped properly to a category

The `isValid()` method checks:

1. If the symbol belongs to a Container
2. If all domain set symbols exist in the Container
3. If all domain set symbols objects are valid
4. If records are a DataFrame (or `None`)
5. The shape of the records is congruent with the dimensionality of the symbol
6. If records column headings are in standard format
7. If all domain columns are type `category` and also ordered
8. If all domain categorical dtypes are referenced properly (`records` for referenced domain sets cannot be `None` in order to create categoricals properly)
9. If there are any domain violations
10. If there are any duplicate domain members
11. That all data columns are type `float`
12. To make sure that all domain categories are type `str`
7.5.2.11 Domain Forwarding

GAMS includes the ability to define sets directly from data using the implicit set notation (see: Implicit Set Definition (or: Domain Defining Symbol Declarations)). This notation has an analogue in GAMS Transfer called \texttt{domain\_forwarding}.

\textbf{Note}

It is possible to recursively update a subset tree in GAMS Transfer.

Domain forwarding is available as an argument to all symbol object constructors; the user would simply need to pass \texttt{domain\_forwarding=True}.

In this example we have raw data that in the dist DataFrame and we want to send the domain information into the i and j sets – we take care to pass the set objects as the domain for parameter c.

\begin{verbatim}
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i")
j = gt.Set(m, "j")
dist = pd.DataFrame(
    ["seattle", "new-york", 2.5],
    ["seattle", "chicago", 1.7],
    ["seattle", "topeka", 1.8],
    ["san-diego", "new-york", 2.5],
    ["san-diego", "chicago", 1.8],
    ["san-diego", "topeka", 1.4],
),
    columns=["from", "to", "thousand_miles"],
) c = gt.Parameter(m, "c", [i, j], records=dist, domain\_forwarding=True)
\end{verbatim}

In \[1\]: i.records
\begin{verbatim}
Out[1]:
  uni_0 element     text
0  seattle
1  san-diego
\end{verbatim}

In \[2\]: j.records
\begin{verbatim}
Out[2]:
  uni_0 element     text
0  new-york
1  chicago
2  topeka
\end{verbatim}

\textbf{Note}

The element order in the sets i and j mirrors that in the raw data.

In this example we show that domain forwarding will also work recursively to update the entire set lineage – the domain forwarding occurs at the creation of every symbol object. The correct order of elements in set i is \([z, a, b, c]\) because the records from j are forwarded first, and then the records from k are propagated through (back to i).

\begin{verbatim}
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i")
j = gt.Set(m, "j", i, records=["z"], domain\_forwarding=True)
k = gt.Set(m, "k", j, records=["a", "b", "c"], domain\_forwarding=True)
\end{verbatim}

In \[1\]: i.records
\begin{verbatim}
Out[1]:
  uni_0 element     text
0  z
1  a
2  b
3  c
\end{verbatim}

In \[2\]: j.records
7.5 Python API

Out[2]:
[0x0] 0 element_text
[0x0] 0 z
[0x0] 1 a
[0x0] 2 b
[0x0] 3 c

In [3]: k.records
Out[3]:
[0x0] 0 element_text
[0x0] 0 a
[0x0] 1 b
[0x0] 2 c

7.5.2.12 Describing Data

The methods `describeSets`, `describeParameters`, `describeVariables`, and `describeEquations` allow the user to get a summary view of key data statistics. The returned DataFrame aggregates the output for a number of other methods (depending on symbol type). A description of each `Container` method is provided in the following subsections:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, str, NoneType</td>
<td>A list of sets in the <code>Container</code> to include in the output. <code>describeSets</code> will include aliases if they are explicitly passed by the user.</td>
<td>No</td>
<td>None (if None specified, will assume all sets – not aliases)</td>
</tr>
</tbody>
</table>

7.5.2.12.1 describeSets Returns: pandas.DataFrame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Property / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>is_singleton</td>
<td>bool if the set/alias is a singleton set (or an alias of a singleton set)</td>
</tr>
<tr>
<td>alias_with</td>
<td>[OPTIONAL if users passes an alias name as part of symbols] name of the parent set (for alias only), None otherwise</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>num_recs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>$1 - \text{num_recs/cardinality}$</td>
</tr>
</tbody>
</table>

Example #1

```python
import gamstransfer as gt
m = gt.Container("transport.gdx")
In [1]: m.describeSets()
Out[1]:
<table>
<thead>
<tr>
<th>name</th>
<th>is_singleton</th>
<th>domain_type</th>
<th>dim</th>
<th>num_recs</th>
<th>cardinality</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>False</td>
<td>[*]</td>
<td>none</td>
<td>1</td>
<td>2</td>
<td>None</td>
</tr>
<tr>
<td>j</td>
<td>False</td>
<td>[*]</td>
<td>none</td>
<td>1</td>
<td>3</td>
<td>None</td>
</tr>
</tbody>
</table>
```
Example #2 – with aliases

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=['i' + str(i) for i in range(1, 10)])
j = gt.Set(m, "j", records=['j' + str(i) for i in range(1, 10)])
ip = gt.Alias(m, "ip", i)
jp = gt.Alias(m, "jp", j)
```

In [1]: m.describeSets()
Out[1]:
```
<table>
<thead>
<tr>
<th>name</th>
<th>is_singleton</th>
<th>domain</th>
<th>domain_type</th>
<th>dim</th>
<th>num_recs</th>
<th>cardinality</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>0</td>
<td>9</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>j</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>0</td>
<td>9</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
```

In [2]: m.describeSets(m.listSets() + m.listAliases())
Out[2]:
```
<table>
<thead>
<tr>
<th>name</th>
<th>is_singleton</th>
<th>is_alias</th>
<th>alias</th>
<th>with_domain</th>
<th>domain</th>
<th>domain_type</th>
<th>dim</th>
<th>num_recs</th>
<th>cardinality</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>False</td>
<td>True</td>
<td>ip</td>
<td>False</td>
<td>i</td>
<td>False</td>
<td>0</td>
<td>9</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>j</td>
<td>False</td>
<td>False</td>
<td>None</td>
<td>False</td>
<td>None</td>
<td>False</td>
<td>0</td>
<td>9</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>jp</td>
<td>False</td>
<td>True</td>
<td>jp</td>
<td>False</td>
<td>j</td>
<td>False</td>
<td>0</td>
<td>9</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, str, NoneType</td>
<td>A list of parameters in the Container to include in the output</td>
<td>No</td>
<td>None (if None specified, will assume all parameters)</td>
</tr>
</tbody>
</table>

7.5.2.12.2 describeParameters Returns: pandas.DataFrame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Property / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>is_scalar</td>
<td>bool if the symbol is a scalar (i.e., dimension = 0)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>num_recs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>min_value</td>
<td>min value in data</td>
</tr>
<tr>
<td>mean_value</td>
<td>mean value in data</td>
</tr>
<tr>
<td>max_value</td>
<td>max value in data</td>
</tr>
<tr>
<td>where_min</td>
<td>domain of min value (if multiple, returns only first occurrence)</td>
</tr>
<tr>
<td>where_max</td>
<td>domain of max value (if multiple, returns only first occurrence)</td>
</tr>
<tr>
<td>count_eps</td>
<td>number of SpecialValues.EPS in data</td>
</tr>
<tr>
<td>count_na</td>
<td>number of SpecialValues.NA in data</td>
</tr>
<tr>
<td>count_undef</td>
<td>number of SpecialValues.UNDEF in data</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - num_recs/cardinality</td>
</tr>
</tbody>
</table>

Example

```python
import gamstransfer as gt
m = gt.Container("transp.gdx")
In [1]: m.describeParameters()
Out[1]:
```
```
| name | is_scalar | domain | domain_type | dim | num_recs | min_value | mean_value | max_value | where_min | count_eps | count_na | count_undef | cardinality | sparsity |
|------|-----------|--------|-------------|-----|----------|-----------|------------|-----------|-----------|-----------|-----------|-----------|-------------|------------|----------|
| i    | False     | False  | False       | 0   | 9        | None      | None       | None      | None      | None      | None      | None      | 1 - num_recs/cardinality |
```
```
7.5 Python API

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, str, NoneType</td>
<td>A list of variables in the Container to include in the output</td>
<td>No</td>
<td>None (if None specified, will assume all variables)</td>
</tr>
</tbody>
</table>

7.5.2.12.3 describeVariables  Returns: pandas.DataFrame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Property / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>type</td>
<td>type of variable (i.e., binary, integer, positive, negative, free, sos1, sos2, semicont, semiint)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>num_recs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - num_recs/cardinality</td>
</tr>
<tr>
<td>min_level</td>
<td>min value in the level</td>
</tr>
<tr>
<td>mean_level</td>
<td>mean value in the level</td>
</tr>
<tr>
<td>max_level</td>
<td>max value in the level</td>
</tr>
<tr>
<td>where_max_abs_level</td>
<td>domain of max(abs(level)) in data</td>
</tr>
<tr>
<td>count_eps_level</td>
<td>number of SpecialValues.EPS in level</td>
</tr>
<tr>
<td>min_marginal</td>
<td>min value in the marginal</td>
</tr>
<tr>
<td>mean_marginal</td>
<td>mean value in the marginal</td>
</tr>
<tr>
<td>max_marginal</td>
<td>max value in the marginal</td>
</tr>
<tr>
<td>where_max_abs_marginal</td>
<td>domain of max(abs(marginal)) in data</td>
</tr>
<tr>
<td>count_eps_marginal</td>
<td>number of SpecialValues.EPS in marginal</td>
</tr>
</tbody>
</table>

Example

```python
import gamstransfer as gt
m = gt.Container("trnsport.gdx")
m.describeVariables()
```

<table>
<thead>
<tr>
<th></th>
<th>type</th>
<th>domain</th>
<th>domain_type</th>
<th>dim</th>
<th>num_recs</th>
<th>cardinality</th>
<th>sparsity</th>
<th>min_level</th>
<th>mean_level</th>
<th>max_level</th>
<th>where_max_abs_level</th>
<th>count_eps_level</th>
<th>min_marginal</th>
<th>mean_marginal</th>
<th>max_marginal</th>
<th>where_max_abs_marginal</th>
<th>count_eps_marginal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>positive</td>
<td>[i, j]</td>
<td>2</td>
<td>6</td>
<td>0.008</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>z</td>
<td>free</td>
<td>[]</td>
<td>0</td>
<td>0.0</td>
<td>0.000</td>
<td>0.036</td>
<td>153.675</td>
<td>163.675</td>
<td>153.675</td>
<td>153.675</td>
<td>153.675</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
7.5.2.12.4 describeEquations  Returns: pandas.DataFrame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Property / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>type</td>
<td>type of variable (i.e., binary, integer, positive, negative, free, sos1, sos2, semicont, semint)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>num_recs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - num_recs/cardinality</td>
</tr>
<tr>
<td>min_level</td>
<td>min value in the level</td>
</tr>
<tr>
<td>mean_level</td>
<td>mean value in the level</td>
</tr>
<tr>
<td>max_level</td>
<td>max value in the level</td>
</tr>
<tr>
<td>where_max_abs_level</td>
<td>domain of max(abs(level)) in data</td>
</tr>
<tr>
<td>count_eps_level</td>
<td>number of SpecialValues.EPS in level</td>
</tr>
<tr>
<td>min_marginal</td>
<td>min value in the marginal</td>
</tr>
<tr>
<td>mean_marginal</td>
<td>mean value in the marginal</td>
</tr>
<tr>
<td>max_marginal</td>
<td>max value in the marginal</td>
</tr>
<tr>
<td>where_max_abs_marginal</td>
<td>domain of max(abs(marginal)) in data</td>
</tr>
<tr>
<td>count_eps_marginal</td>
<td>number of SpecialValues.EPS in marginal</td>
</tr>
</tbody>
</table>

Example

```python
import gamstransfer as gt
m = gt.Container("transport.gdx")
In [1]: m.describeEquations()
Out[1]:
name     type  domain  domain_type  dim  num_recs  cardinality  sparsity  min_level  mean_level  max_level  where_max_abs_level  count_eps_level  min_marginal  mean_marginal  max_marginal  where_max_abs_marginal  count_eps_marginal
0  cost  eq          []  none     0  1.000  1.000  None  None  -0.0  0.0  -0.0  0  0  0  0  0  0
1  demand  eq  [j]  regular  1  1.000  1.000  3  3  0.0  275.0  300.0  325.0  0  0.126  0.168  0.225  [new-york]  0
2  supply  leq  [i]  regular  1  1.000  1.000  2  2  0.0  350.0  450.0  550.0  [san-diego]  0  0.000  0.000  0.000  [seattle]  1
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, str, NoneType</td>
<td>A list of alias (only) symbols in the Container to include in the output</td>
<td>No</td>
<td>None (if None specified, will assume all aliases – not sets)</td>
</tr>
</tbody>
</table>
7.5 Python API

7.5.2.12.5 describeAliases  
Returns: pandas.DataFrame

The following table includes a short description of the column headings in the return. All data is referenced from the parent set that the alias is created from.

<table>
<thead>
<tr>
<th>Property / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>is Singleton</td>
<td>bool if the set/alias is a singleton set (or an alias of a singleton set)</td>
</tr>
<tr>
<td>alias_with</td>
<td>name of the parent set (for alias only), None otherwise</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>num_recs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>$1 - \text{num_recs}/\text{cardinality}$</td>
</tr>
</tbody>
</table>

Example

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", records=["j" + str(j) for j in range(10)])
ip = gt.Alias(m, "ip", i)
ipp = gt.Alias(m, "ipp", ip)
jp = gt.Alias(m, "jp", j)
In [1]: m.describeAliases()
Out[1]:
    name alias_with is Singleton domain domain_type dim num_recs cardinality sparsity
 0     ip    i False [*] none 1 5      None      None
 1    ipp   i False [*] none 1 5      None      None
 2      jp   j False [*] none 1 10     None      None
```

7.5.2.13 Matrix Generation

GAMS Transfer stores data in a "flat" format, that is, one record entry per DataFrame row. However, it is often necessary to convert this data format into a matrix format – GAMS Transfer enables users to do this with relative ease using the `toDense` and the `toSparseCoo` symbol methods. The `toDense` method will return a dense N-dimensional numpy array with each dimension corresponding to the GAMS symbol dimension; it is possible to output an array up to 20 dimensions (a GAMS limit). The `toSparseCoo` method will return the data in a sparse scipy COOrdinate format, which can then be efficiently converted into other sparse matrix formats.

Attention

Both the `toDense` and `toSparseCoo` methods do not transform the underlying DataFrame in any way, they only return the transformed data.

Note

`toSparseCoo` will only convert 2-dimensional data to the scipy COOrdinate format. A user interested in sparse data for an N-dimensional symbol will need to decide how to reshape the dense array in order to generate the 2D sparse format.
Attention

In order to use the `toSparseCoo` method the user will need to install the scipy package. Scipy is not provided with GMSPython.

Both the `toDense` and `toSparseCoo` method leverage the indexing that comes along with using categorical data types to store domain information. This means that linking symbols together (by passing symbol objects as domain information) impacts the size of the matrix. This is best demonstrated by a few examples.

Example (1D data w/o domain linking (i.e., a relaxed domain))

```python
import gamstransfer as gt
m = gt.Container()
a = gt.Parameter(m, "a", "i", records=[("a", 1), ("c", 3)])
In [1]: a.records
Out[1]:
   i  value
0  a   1.0
1  c   3.0
In [2]: a.toDense()
Out[2]: array([1., 3.])
In [3]: a.toSparseCoo()
Out[3]:
<1x2 sparse matrix of type '<class 'numpy.float64'>'
    with 2 stored elements in COOrdinate format>
```

Note that the parameter `a` is not linked to another symbol, so when converting to a matrix, the indexing is referenced to the data structure in `a.records`. Defining a sparse parameter `a` over a set `i` allows us to extract information from the `i` domain and construct a very different dense matrix, as the following example shows:

Example (1D data w/ domain linking (i.e., a regular domain))

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c", "d"])
a = gt.Parameter(m, "a", i, records=[("a", 1), ("c", 3)])
In [1]: i.records
Out[1]:
   uni 0 element_text
0   a
1   b
2   c
3   d
In [2]: a.records
Out[2]:
   i  value
0  a   1.0
1  c   3.0
In [3]: a.toDense()
Out[3]: array([1., 0., 3., 0.])
In [4]: a.toSparseCoo()
Out[4]:
<1x4 sparse matrix of type '<class 'numpy.float64'>'
    with 2 stored elements in COOrdinate format>
```

Example (2D data w/ domain linking)

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c", "d"])
a = gt.Parameter(m, "a", [i, i], records=[("a", "a", 1), ("c", "c", 3)])
In [1]: i.records
Out[1]:
   uni 0 element_text
0   a
```
7.5.2.14 The Universe Set

A Unique Element List (UEL) (aka the "universe" or "universe set") is an \((i, s)\) pair where \(i\) is an identification number for a string \(s\). GAMS uses UELs to efficiently store domain entries of a record by storing the UEL ID \(i\) of a domain entry instead of the actual string \(s\). This avoids storing the same string multiple times. The concept of UELs also exists in Python/Pandas and is called a "categorical array". GAMS Transfer leverages these types in order to efficiently store strings and enable domain checking within the Python environment.

Each domain column in a DataFrame can be assigned a unique categorical type, the effect is that each symbol maintains its own list of UELs per dimension. It is possible to convert a categorical column to its ID number representation by using the categorical accessor \(x\.records[[\text{domain column label}]].cat\.codes\); however, this type of data manipulation is not necessary within GAMS Transfer, but could be handy when debugging data.

Pandas offers the possibility to create categorical column types that are \textit{ordered} or not; GAMS Transfer relies exclusively on \textit{ordered} categorical data types (in order for a symbol to be valid it must have only \textit{ordered} categories). By using ordered categories, GAMS Transfer will order the UEL such that elements appear in the order in which they appeared in the data (which is how GAMS defines the UEL). \texttt{GAMSTransfer} allows the user to reorder the UELs with the \texttt{uel优先} argument in the \texttt{.write()} method.

GAMS Transfer does not actually keep track of the UEL separately from other symbols in the Container, it will be created internal to the \texttt{.write()} method and is based on the order in which data is added to the container. The user can access the current state of the UEL with the \texttt{.getUniverseSet()} container method. For example, we set a two dimensional set:

```python
import gamstransfer as gt
m = gt.Container()
j = gt.Set(m, "j", ["*", "]", records=[("i" + str(n), "]" + str(n)) for n in range(2)])
In [1]: j.records
Out[1]: [[uni0, uni1], element_text
0 10 10
1 11 11]
```

Pandas also includes a number of methods that allow categories to be \textit{renamed}, \textit{appended}, etc. These methods may be useful for advanced users, but most users will probably find that modifying the original data structures and resetting the symbol records provides a simpler solution. The design of GAMS Transfer should enable the user to quickly move data back and forth, without worrying about the deeper mechanics of categorical data.
7.5.2.15 Reordering Symbols

The order of the Container file requires the symbols to be sorted such that, for example, a Set used as domain of another symbol appears before that symbol. The Container will try to establish a valid ordering when writing the data. This type of situation could be encountered if the user is adding and removing many symbols (and perhaps rewriting symbols with the same name) — users should attempt to only add symbols to a Container once, and care must be taken when creating symbol names. The method reorderSymbols attempts to fix symbol ordering problems. The following example shows how this can occur:

Example Symbol reordering

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", i, records=["i" + str(i) for i in range(3)])
In [1]: m.data
Out[1]:
{'i': <src.gamstransfer.Set at 0x7f90c068a8e0>,
 'j': <src.gamstransfer.Set at 0x7f908084ceb0>}

# now we remove the set i and recreate the data
m.removeSymbols("i")
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
In [1]: m.data
Out[1]:
{'j': <src.gamstransfer.Set at 0x7f90c068a8e0>,
 'i': <src.gamstransfer.Set at 0x7f908084ceb0>}

# calling reorderSymbols() will order the dictionary properly, but the domain reference in j is now broken
m.reorderSymbols()

# fix the domain reference in the set j
j.domain = i
In [1]: m.isValid()
Out[1]: True
```

7.5.2.16 Rename Symbols

It is possible to rename a symbol even after it has been added to a Container. There are two methods that can be used to achieve the desired outcome:

- using the container method renameSymbol
- directly changing the name symbol property

We create a Container with two sets:

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"])
```

Example #1 - Change the name of a symbol with the container method

```python
In [1]: m.renameSymbol("i", 'h')
In [2]: m.data
Out[2]:
{'j': <src.gamstransfer.Set at 0x7f9f00f0e70>,
 'h': <src.gamstransfer.Set at 0x7f9f009e2b0>}
```
Example #2 - Change the name of a symbol with the .name attribute

```
In [1]: i.name = 'h'
In [2]: m.data
Out[2]:
{'h': <src.gamstransfer.Set at 0x7f9fc01fc070>,
    'j': <src.gamstransfer.Set at 0x7f9f9080a220>}
```

**Note**

Note that the renamed symbols maintain the original symbol order; this will prevent unnecessary reordering operations later in the workflow.

### 7.5.2.17 Removing Symbols

Removing symbols from a container is easy when using the `removeSymbols` container method; this method accepts either a `str` or a `list` of `str`.

**Attention**

Once a symbol has been removed, it is possible to have hanging references as domain links in other symbols. The user will need to repair these other symbols with the proper domain links in order to avoid validity errors.

### 7.5.2.18 Full Example

It is possible to use everything we now know about GAMS Transfer to recreate the `transport.gms` results in GDX form. As part of this example we also introduce the `write` method (and generate `new.gdx`). We will discuss it in more detail in the following section: Data Exchange with GDX.

```python
import gamstransfer as gt

# create an empty Container object
m = gt.Container()

# add sets
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"], description="markets")

# add parameters
a = gt.Parameter(m, "a", ["*"], description="capacity of plant i in cases")
b = gt.Parameter(m, "b", ["j"], description="demand at market j in cases")
d = gt.Parameter(m, "d", ["i", "j"], description="distance in thousands of miles")
f = gt.Parameter(m, "f", ["i", "j"], description="freight in dollars per case per thousand miles")
c = gt.Parameter(m, "c", ["i", "j"], description="transport cost in thousands of dollars per case")

# set parameter records
cap = pd.DataFrame([("seattle", 350), ("san-diego", 600)], columns=["plant", "n_cases"])
a.setRecords(cap)
dem = pd.DataFrame([("new-york", 325), ("chicago", 300), ("topeka", 275)], columns=["market", "n_cases"])
b.setRecords(dem)
d.setRecords(dist)

cost = d.records.copy(deep=True)
cost["value"] = f.records.loc[0, "value"] * cost["value"] / 1000
c.setRecords(cost)
```
# add variables
q = pd.DataFrame(
    [
        ("seattle", "new-york", 50, 0),
        ("seattle", "chicago", 300, 0),
        ("seattle", "topeka", 0, 0.036),
        ("san-diego", "new-york", 275, 0),
        ("san-diego", "chicago", 0, 0.009),
        ("san-diego", "topeka", 275, 0),
    ],
    columns=["from", "to", "level", "marginal"],
)

# add equations
cost = gt.Equation(m, "cost", "eq", description="define objective function")
supply = gt.Equation(m, "supply", "leq", i, description="observe supply limit at plant i")
demand = gt.Equation(m, "demand", "geq", j, description="satisfy demand at market j")

7.5.2.19 Advanced Examples

With the wide range of I/O tools included in Pandas it is possible to easily draw data down from CSV
data sources. We provide two examples here that pull data into GAMS Transfer from an HTML source and
a POSTGRES SQL source.

Example #1 - Create symbols from HTML

```python
import gamstransfer as gt
import pandas as pd
url = "https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list"
dfs = pd.read_html(url)
# pandas will create a list of dataframe depending on the target URL, we just need the first one
df = dfs[0]
m = gt.Container()
b = gt.Set(m, "b", ["*"] , records=df["Bank Name"].unique(), description="Bank Name")
s = gt.Set(m, "s", ["*"] , records=df["StateSt"].sort_values().unique(), description="States (alphabetical order)")
c = gt.Set(m, "c", ["*"] , records=df["CityCity"].sort_values().unique(), description="Cities (alphabetical order)")
```
7.5 Python API

\[\text{c_to_s} = \text{gt.Set}(\text{m}, \text{"c_to_s"}, \text{[c, s]}, \text{records} = \text{df[\"CityCity", \"StateSt\"]} \text{.drop}_{\text{duplicates}}() \text{.sort}_{\text{values}}(\text{by} = \text{[\"StateSt", \"CityCity\"]}, \text{description} = \text{\"City/State pair\")} \]

\[\text{bf} = \text{gt.Parameter}(\text{m}, \text{"bf"}, \text{b}, \text{records} = \text{df[\"Bank NameBank", \"FundFund\"]} \text{.drop}_{\text{duplicates}}(\text{subset} = \text{\"Bank NameBank\")} \text{.sort}_{\text{values}}(\text{by} = \text{\"Bank NameBank\")}, \text{description} = \text{\"Bank Name & Fund "} \]

\text{In [1]: m.isValid()} 
\text{Out[1]: True}

\text{Note}

Users can chain Pandas operations together and pass those operations through to the records argument or the setRecords method.

Example #2 - Create symbols from a POSTGRES SQL Database (sqlalchemy)

```
import gamstransfer as gt
from sqlalchemy import create_engine
import pandas as pd

# connect to postgres (assuming a localhost)
engine = create_engine("postgresql://localhost:5432/" + <database_name>)
df = pd.read_sql(<sql_table_name>, con=engine, index_col=0)

# create the Container and add symbol
m = Container()
p = Parameter(m, <sql_table_name>)

# we need to figure out the symbol dimensionality (potentially from the shape of the dataframe)
r, c = df.shape
p.dimension = c - 1

# set the records
p.setRecords(df)

# write out the GDX file
m.write("out.gdx")
```

7.5.2.20 GAMS Special Values

The GAMS system contains five special values: \text{UNDEF} (undefined), \text{NA} (not available), \text{EPS} (epsilon), \text{+INF} (positive infinity), \text{-INF} (negative infinity). These special values must be mapped to their Python equivalents. GAMS Transfer follows the following convention to generate the 1:1 mapping:

- \text{+INF} is mapped to \text{float("inf")}
- \text{-INF} is mapped to \text{float("-inf")}
- \text{EPS} is mapped to \text{-0.0} (mathematically identical to zero)
- \text{NA} is mapped to a special \text{NaN}
- \text{UNDEF} is mapped to \text{float("nan")}

GAMS Transfer syntax is designed to quickly get data into a form that is usable in further analyses or visualization; this mapping also highlights the preference for data that is of type \text{float}, which offers performance benefits within Pandas/NumPy. The user does not need to remember these constants as they are provided within the class \text{SpecialValues} as \text{SpecialValues.POSINF}, \text{SpecialValues.NEGINF}, \text{SpecialValues.EPS}, \text{SpecialValues.NA}, and \text{SpecialValues.UNDEF}. The \text{SpecialValues} class also contains methods to test for these special values. Some examples are shown below; already, we, begin to introduce some of the GAMS Transfer syntax.
Example (special values in a parameter)

```python
import gamstransfer as gt
m = gt.Container()
x = gt.Parameter(
    m,
    "x",
    ["*"],
    records=[
        ('i1', 1),
        ('i2', SpecialValues.POSINF),
        ('i3', SpecialValues.NEGINF),
        ('i4', SpecialValues.EPS),
        ('i5', SpecialValues.NA),
        ('i6', SpecialValues.UNDEF),
    ],
    description="special values",
)
```

The following DataFrame for `x` would look like:

```
In [1]: x.records
Out[1]:
       value
unl 0   1.0
   1   inf
   2  -inf
   3   -0.0
   4    NaN
   5    NaN
```

The user can now easily test for specific special values in the `value` column of the DataFrame (returns a boolean array):

```python
In [1]: SpecialValues.isNA(x.records['value'])
Out[1]: array([False, False, False, False, True, False])
```

Other data structures can be passed into these methods as long as these structures can be converted into a numpy array with `dtype=float`. It follows that:

```python
In [1]: SpecialValues.isEps(SpecialValues.EPS)
Out[1]: True
In [2]: SpecialValues.isPosInf(SpecialValues.POSINF)
Out[2]: True
In [3]: SpecialValues.isNegInf(SpecialValues.NEGINF)
Out[3]: True
In [4]: SpecialValues.isNA(SpecialValues.NA)
Out[4]: True
In [5]: SpecialValues.isUndef(SpecialValues.UNDEF)
Out[5]: True
In [6]: SpecialValues.isNA(SpecialValues.UNDEF)
Out[6]: False
```

Pandas DataFrames allow data columns to exist with mixed type (`dtype=object`) – GAMS Transfer leverages this convenience feature to enable users to import string representations of `EPS`, `NA`, and `UNDEF`. GAMS Transfer is tolerant of any mixed-case special value string representation. Python offers additional flexibility when representing negative/positive infinity. Any string `x` where `float(x) == float("inf")` evaluates to True can be used to represent positive infinity. Similarly, any string `x` where `float(x) == float("-inf")` evaluates to True can be used to represent negative infinity. Allowed values include `inf`, `+inf`, `INFINITY`, `+INFINITY`, `-inf`, `-INFINITY` and all mixed-case equivalents.

Example (special values defined by strings)

```python
import gamstransfer as gt
m = gt.Container()
x = gt.Parameter(
    m,
    "x",
    ["*"],
    records=[
        ('i1', 1),
        ('i2', "+inf"),
        ('i3', "-infinity"),
    ],
    description="special values",
)```
These special strings will be immediately mapped to their float equivalents from the `SpecialValues` class in order to ensure that all data entries are float types.

### 7.5.2.21 GAMS Transfer Standard Data Formats

This section is meant to introduce the standard format that GAMS Transfer expects for symbol records. It has already been mentioned that we store data as a Pandas DataFrame, but there is an assumed structure to the column headings and column types that will be important to understand. GAMS Transfer includes convenience functions in order to ease the burden of converting data from a user-centric format to one that is understood by GAMS Transfer. However, advanced users will want to convert their data first and add it directly to the Container to avoid making extra copies of (potentially large) data sets.

**Set Records Standard Format**

All set records (including singleton sets) are stored as a Pandas DataFrame with `n` number of columns, where `n` is the dimensionality of the symbol + 1. The first `n-1` columns include the domain elements while the last column includes the set element explanatory text. Records are organized such that there is one record per row.

The names of the domain columns follow a pattern of `<set_name>_<index_position>;` a symbol dimension that is referenced to the universe is labeled `uni_<index_position>`. The explanatory text column is called `element_text` and must take the last position in the DataFrame.

All domain columns must be a categorical data type and the `element_text` column must be a `object` type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however GAMS Transfer requires that all these are type `str`. All rows in the `element_text` column must be type `str`.

Some examples:

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
j = gt.Set(m, "j", [i, "*"], records=[("seattle", "new-york"), ("san-diego", "st-louis")])
k = gt.Set(m, "k", [i], is_singleton=True, records=["seattle"])
In [1]: i.records
Out[1]:
   uni_0   element_text
0 seattle
1 san-diego
In [2]: j.records
Out[2]:
    uni_1   i_0   element_text
0 seattle new-york
1 san-diego st-louis
In [3]: k.records
Out[3]:
   uni_0   element_text
0 seattle
```


Parameter Records Standard Format

All parameter records (including scalars) are stored as a Pandas DataFrame with $n$ number of columns, where $n$ is the dimensionality of the symbol + 1. The first $n-1$ columns include the domain elements while the last column includes the numerical value of the records. Records are organized such that there is one record per row. Scalar parameters have zero dimension, therefore they only have one column and one row.

The names of the domain columns follow a pattern of $\text{<set\_name>\_<index\_position>}$: a symbol dimension that is referenced to the universe is labeled $\text{uni\_<index\_position>}$). The value column is called $\text{value}$ and must take the last position in the DataFrame.

All domain columns must be a categorical data type and the $\text{value}$ column must be a float type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however GAMS Transfer requires that all these are type str.

Some examples:

```python
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
a = gt.Parameter(m, "a", ["*"], records=[("seattle", 50), ("san-diego", 100)])
b = gt.Parameter(m, "b", [i, "*"], records=[("seattle", "new-york", 32.2), ("san-diego", "st-louis", 123)],
)  
c = gt.Parameter(m, "c", records=90)
In [1]: a.records
Out[1]:
  uni_0 value
 0 seattle 50.0
 1 san-diego 100.0
In [2]: b.records
Out[2]:
  i_0 uni_1 value
 0 seattle new-york 32.2
 1 san-diego st-louis 123.0
In [3]: c.records
Out[3]:
  value
 0 90.0
```

Variable/Equation Records Standard Format

Variables and equations share the same standard data format. All records (including scalar variables/equations) are stored as a Pandas DataFrame with $n$ number of columns, where $n$ is the dimensionality of the symbol + 5. The first $n-5$ columns include the domain elements while the last five columns include the numerical values for different attributes of the records. Records are organized such that there is one record per row. Scalar variables/equations have zero dimension, therefore they have five columns and one row.

The names of the domain columns follow a pattern of $\text{<set\_name>\_<index\_position>}$: a symbol dimension that is referenced to the universe is labeled $\text{uni\_<index\_position>}$). The attribute columns are called $\text{level}$, $\text{marginal}$, $\text{lower}$, $\text{upper}$, and $\text{scale}$. These attribute columns must appear in this order. Attributes that are not supplied by the user will be assigned the default GAMS values for that variable/equation type; it is possible to not pass any attributes, GAMS Transfer would then simply assign default values to all attributes.

All domain columns must be a categorical data type and all the attribute columns must be a float type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however GAMS Transfer requires that all these are type str.
Some examples:

```python
import gamstransfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
a = gt.Variable(m, "a", "free", domain=[i], records=pd.DataFrame(
    ["seattle", 50], ["san-diego", 100]), columns=["city", "level"]
).
In [1]: a.records
Out[1]:
   city     level marginal lower  upper  scale
   0  seattle 50.0 0.0  -inf  inf  1.0
   1 san-diego 100.0 0.0  -inf  inf  1.0
```

### 7.5.2.22 Data Exchange with GDX

Up until now, we have been focused on using GAMS Transfer to create symbols in an empty `Container` using the symbol constructors (or their corresponding container methods). These tools will enable users to ingest data from many different formats and add them to a `Container` – however, it is also possible to read in symbol data directly from GDX files using the `read` container method. In the following sections, we will discuss this method in detail as well as the `write` method, which allows users to write out to new GDX files.

#### 7.5.2.22.1 Reading from GDX

There are two main ways to read in GDX based data.

- Pass the file path directly to the `Container` constructor (will read all symbols and records)
- Pass the file path directly to the `read` method (default read all symbols, but can read partial files)

The first option here is provided for convenience and will, internally, call the `read` method. This method will read in all symbols as well as their records. This is the easiest and fastest way to get data out of a GDX file and into your Python environment. For the following examples we leverage the GDX output generated from the `trnsport.gms` model file.

Example (reading full data w/ `Container` constructor)

```python
import gamstransfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.data
Out[1]:

```

A user could also read in data with the `read` method as shown in the following example.
Example (reading full data w/ read method)

```python
import gamstransfer as gt
m = gt.Container()
m.read("transport.gdx")
```

```
In [1]: m.data
Out[1]:
{
    'i': <src.gamstransfer.Set at 0x7fdd21858d60>,
    'j': <src.gamstransfer.Set at 0x7fdd21858dc0>,
    'a': <src.gamstransfer.Parameter at 0x7fdd21858df0>,
    'b': <src.gamstransfer.Parameter at 0x7fdd21858d90>,
    'd': <src.gamstransfer.Parameter at 0x7fdd21858e80>,
    'f': <src.gamstransfer.Parameter at 0x7fdd21858eb0>,
    'c': <src.gamstransfer.Parameter at 0x7fdd21858ee0>,
    'x': <src.gamstransfer.Variable at 0x7fdd21858f10>,
    'z': <src.gamstransfer.Variable at 0x7fdd21858e50>,
    'cost': <src.gamstransfer.Equation at 0x7fdd21858f70>,
    'supply': <src.gamstransfer.Equation at 0x7fdd21858fa0>,
    'demand': <src.gamstransfer.Equation at 0x7fdd21858fd0>
}
```

It is also possible to read in a partial GDX file with the `read` method, as shown in the following example:

```python
m = gt.Container()
m.read("transport.gdx", "x")
```

```
In [1]: m.data
Out[1]:
{
    'x': <src.gamstransfer.Variable at 0x7fa728a2d9d0>
}
```

```
In [2]: m.data['x'].records
Out[2]:
0 seattle new-york 50.0 0.000 0.0 inf 1.0
1 seattle chicago 300.0 0.000 0.0 inf 1.0
2 seattle topeka 0.0 0.036 0.0 inf 1.0
3 san-diego new-york 275.0 0.000 0.0 inf 1.0
4 san-diego chicago 0.0 0.009 0.0 inf 1.0
5 san-diego topeka 275.0 0.000 0.0 inf 1.0
```

This syntax assumes that the user will always want to read in both the metadata as well as the actual data records, but it is possible to skip the reading of the records by passing the argument `records=False`.

```python
m = gt.Container()
m.read("transport.gdx", "x", records=False)
```

```
In [1]: m.data
Out[1]:
{
    'x': <src.gamstransfer.Variable at 0x7fa728a37220>
}
```

```
In [2]: m.data['x'].summary
Out[2]:
{
    'name': 'x',
    'type': 'positive',
    'domain_objects': ['i', 'j'],
    'domain_names': ['i', 'j'],
    'dimension': 2,
    'description': 'shipment quantities in cases',
    'number_records': None,
    'domain_type': 'relaxed'
}
```

```
In [3]: type(m.data['x'].records)
Out[3]: <class 'NoneType'>
```

Attention

The `read` method attempts to link the domain objects together (in order to have a "regular" domain type) but if domain sets are not part of the read operation there is no choice but to default to a "relaxed" domain type. This can be seen in the last example where we only read in the variable x and not the domain sets (i and j) that the variable is defined over. All the data will be available to the user, but domain checking is no longer possible. The symbol x will remain with "relaxed" domain type even if the user were to read in sets i and j in a second `read` call.

7.5.2.22.2 Writing to GDX

A user can write data to a GDX file by simply passing a file path (as a string). The `write` method will then create the GDX and write all data in the `Container`.

Note

It is not possible to write the `Container` when any of its symbols are invalid. If any symbols are invalid an error will be raised and the user will need to inspect the problematic symbols (perhaps using a combination of the `listSymbols(isValid=False)` and `isValid(verbose=True)` methods).
Example

```python
m.write("path/to/file.gdx")
```

Example (write a compressed GDX file)

```python
m.write("path/to/file.gdx", compress=True)
```

Advanced users might want to specify an order to their UEL list (i.e., the universe set); recall that the UEL ordering follows that dictated by the data. As a convenience, it is possible to prepend the UEL list with a user specified order using the `uel_priority` argument.

Example (change the order of the UEL)

```python
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c"])
m.write("foo.gdx", uel_priority=["a", "c"])
```

The original UEL order for this GDX file would have been ["a", "b", "c"], but this example reorders the UEL with `uel_priority` – the positions of b and c have been swapped. This can be verified with the `gdxdump` utility (using the `uelTable` argument):

```
gdxdump foo.gdx uelttable=foo
```

Set foo /
  'a' ,
  'c' ,
  'b' /;
$onEmpty
Set i(*) /
  'a' ,
  'c' ,
  'b' /;
$offEmpty

7.5.2.23 Data Exchange with GamsDatabase and GMD Objects (Embedded Python Code)

We have discussed how to create symbols in an empty `Container` and we have discussed how to exchange data with GDX files, however it is also possible to read and write data directly in memory by interacting with a GamsDatabase/GMD object – this allows GAMS Transfer to be used to read/write data within an Embedded Python Code environment or in combination with the Python OO API. There are some important differences when compared to data exchange with GDX since we are working with data representations in memory.
7.5.2.23.1 Reading from GamsDatabase and GMD Objects  Just as with a GDX, there are two main ways to read in data that is in a GamsDatabase/GMD object.

- Pass the GamsDatabase/GMD object directly to the Container constructor (will read all symbols and records)
- Pass the GamsDatabase/GMD object directly to the `read` method (default read all symbols, but can read partial files)

The first option here is provided for convenience and will, internally, call the `read` method. This method will read in all symbols as well as their records. This is the easiest and fastest way to get data out of a GamsDatabase/GMD object and into your Python environment. While it is possible to generate a custom GamsDatabase/GMD object from scratch (using the gmdcc API), most users will be interacting with a GamsDatabase/GMD object that has already been instantiated internally when he/she is using Embedded Python Code or the GamsDatabase class in the Python OO API. Our examples will show how to access the GamsDatabase/GMD object – we leverage the some of the data from the `transport.gms` model file.

Example (reading full data w/ Container constructor)

```python
m = gt.Container(gams.db)
```

Note

Embedded Python Code users will want pass the GamsDatabase object that is part of the GAMS Database object – this will always be referenced as `gams.db` regardless of the model file.

The following example uses embedded Python code to create a new Container, read in all symbols, and display some summary statistics as part of the gams log output.

```python
Set
   i 'canning plants' / seattle, san-diego /
   j 'markets' / new-york, chicago, topeka /

Parameter
   a(i) 'capacity of plant i in cases'
       / seattle 350
       san-diego 600 /
   b(j) 'demand at market j in cases'
       / new-york 325
       chicago 300
       topeka 275 /

Table d(i,j) 'distance in thousands of miles'
  new-york    chicago    topeka
  seattle 2.5     1.7     1.8
  san-diego 2.5    1.8     1.4;
```

$onembeddedCode Python:
import gamstransfer as gt
m = gt.Container(gams.db)
print(m.describeSets())
print(m.describeParameters())
$offEmbeddedCode
The gams log output will then look as such (the extra print calls are just providing nice spacing for this example):

```
--- Starting compilation
--- matrix.gms(29) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib

<table>
<thead>
<tr>
<th>name</th>
<th>is Singleton</th>
<th>domain</th>
<th>domain type</th>
<th>dim</th>
<th>num_recs</th>
<th>cardinality</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>False</td>
<td>[i]</td>
<td>none</td>
<td>1</td>
<td>2</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>j</td>
<td>False</td>
<td>[j]</td>
<td>none</td>
<td>1</td>
<td>3</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>is scalar</th>
<th>domain</th>
<th>domain type</th>
<th>dim</th>
<th>num_recs</th>
<th>min_value</th>
<th>mean_value</th>
<th>max_value</th>
<th>where</th>
<th>count</th>
<th>cardinality</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>False</td>
<td>[i]</td>
<td>regular</td>
<td>1</td>
<td>2</td>
<td>350.0</td>
<td>475.00</td>
<td>600.0</td>
<td>[seattle]</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>b</td>
<td>False</td>
<td>[j]</td>
<td>regular</td>
<td>1</td>
<td>3</td>
<td>275.0</td>
<td>300.00</td>
<td>325.0</td>
<td>[topeka]</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>d</td>
<td>False</td>
<td>[i, j]</td>
<td>regular</td>
<td>2</td>
<td>6</td>
<td>1.4</td>
<td>1.95</td>
<td>2.5</td>
<td>[san-diego, topeka]</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

--- Starting execution - empty program
*** Status: Normal completion

[3 rows x 16 columns]

--- Starting execution - empty program
*** Status: Normal completion

A user could also read in a subset of the data located in the GamsDatabase object with the read method as shown in the following example. Here we only read in the sets i and j, as a result the .describeParameters() method will return None.

Example (reading subset of full data w/ read method)

Set
  i 'canning plants' / seattle, san-diego /
  j 'markets' / new-york, chicago, topeka /;

Parameter
  a(i) 'capacity of plant i in cases'
    / seattle 350
    san-diego 600 /

  b(j) 'demand at market j in cases'
    / new-york 325
    chicago 300
    topeka 275 /;

Table d(i,j) 'distance in thousands of miles'
  new-york chicago topeka
  seattle 2.5 1.7 1.8
  san-diego 2.5 1.8 1.4;

$onembeddedCode Python:
import gamstransfer as gt
m = gt.Container()
m.read(gams.db, symbols=['i','j'])
```
All the typical functionality of the Container exists when working with GamsDatabase/GMD objects. This means that domain linking, matrix conversion, and other more advanced options are available to the user at either compilation time or execution time (depending on the Embedded Code syntax being used, see: Syntax). The next example generates a 1000x1000 matrix and then takes its inverse using the Numpy linalg package.

Example (Matrix Generation and Inversion)

```plaintext
set i / i1*i1000 /;
alias(i,j);

parameter a(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;
```

```python
import gamstransfer as gt
import numpy as np
import time
gams.printLog("")
s = time.time()
m = gt.Container(gams.db)
gams.printLog(f"read data: {round(time.time() - s, 3)} sec")

s = time.time()
A = m.data["a"].toDense()
gams.printLog(f"create matrix A: {round(time.time() - s, 3)} sec")

s = time.time()
invA = np.linalg.inv(A)
gams.printLog(f"generate inv(A): {round(time.time() - s, 3)} sec")
```
Note

In this example, the assignment of the a parameter is done during execution time so we must use the execution time syntax for embedded code in order to get the numerical records properly.

We will extend this example in the next section to write the inverse matrix A back into a GAMS parameter.

7.5.2.23.2 Writing Data to GamsDatabase and GMD

A user can write to a GamsDatabase/GMD object with the .write() method just as he/she would write a GDX file – however there are some important differences. When a user writes a GDX file the entire GDX file represents a complete data environment (all domains have been resolved, etc.) thus, GAMS Transfer does not need to worry about merge/replace operations. It is possible to merge/replace symbol records when a user is writing data to in-memory data representations with GamsDatabase/GMD. We show a few examples to illustrate this behavior.

Example (Populating a set in GAMS)

* note that we need to declare the set i over "*" in order to provide hints about the symbol dimensionality

set i(*);

$onembeddedCode Python:
import gamstransfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
m.write(gams.db)

$offEmbeddedCode i

embeddedCode Python:
import gamstransfer as gt

m = gt.Container(gams.db)
gams.printLog(""
print(m.data["i"].records)

eendEmbeddedCode
Note

In general, it is possible to use GAMS Transfer to create new symbols in a GamsDatabase and GMD object (and not necessarily merge symbols) but embedded code best practices necessitate the declaration of any GAMS symbols on the GAMS side first, then the records can be filled with GAMS Transfer.

If we break down this example we can see that the set i is declared within GAMS (with no records) and then the records for i are set by writing a Container to the gams.db GamsDatabase object (we do this at compile time). The second embedded Python code block runs at execution time and is simply there to read all the records on the set i—printing the sets this way adds the output to the .log file (we could also use the more common display i; operation in GAMS to display the set elements in the LST file).

Example (Merging set records)

```plaintext
set i / i1, i2 /;

$onmulti
$onembeddedCode Python:
import gamstransfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
m.write(gams.db, merge_symbols="i")

$offEmbeddedCode i
$offmulti

embeddedCode Python:
import gamstransfer as gt
```
In this example we need to make use of $\texttt{onMulti}/\texttt{offMulti}$ in order to merge new set elements into the set $i$ (the same would be true if we were merging other symbol types) – any symbol that already has records defined (in GAMS) and is being added to with Python (and GAMS Transfer) must be wrapped with $\texttt{onMulti}/\texttt{offMulti}$. As with the previous example, the second embedded Python code block runs at execution time and is simply there to read all the records on the set $i$. Note that the UEL order will be different in this case ($i_1$ and $i_2$ come before $i_0$).

Example (Replacing set records)

```plaintext
set i / x1, x2 /;

$onmultiR
$onembeddedCode Python:
import gamstransfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=['"i"'+str(i) for i in range(10)])
m.write(gams.db)
$offEmbeddedCode i
$offmulti
```
embeddedCode Python:
import gamstransfer as gt
m = gt.Container(gams.db)
gams.printLog(""
print(m.data["i"]).records)
endEmbeddedCode

In this example we want to replace the \textit{x1} and \textit{x2} set elements and built up a totally new element list with set elements from the \texttt{Container}. Instead of $\texttt{onMulti/\textbackslash onMulti}$ we must use $\texttt{onMultiR/\textbackslash offMulti}$ to ensure that the replacement happens in GAMS; we also need to remove the set \texttt{i} from the \texttt{merge\_symbols} argument.

\textbf{Attention}

If the user seeks to replace all records in a symbol they must use the $\texttt{onMultiR}$ syntax. It is not sufficient to simply remove them from the \texttt{merge\_symbols} argument in GAMS Transfer. If the user mistakenly uses $\texttt{onMulti}$ the symbols will end up merging without total replacement.

GAMS 38.1.0 Copyright (C) 1987–2022 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(11) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- test.gms(21) 3 Mb
--- Starting execution: elapsed 0:00:01.482
--- test.gms(14) 4 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- uni_0 element_text
  0  i0
  1  i1
  2  i2
  3  i3
  4  i4
  5  i5
  6  i6
  7  i7
  8  i8
  9  i9

*** Status: Normal completion

Example (Merging parameter records)

set i;
parameter a(i<) /
i1 1.23
i2 5
/;
$onmulti
In this example we also need to make use of $onMulti/$offMulti in order to merge new set elements into the set \( i \), however the set \( i \) also needs to contain the elements that are defined in the parameter – here we make use of the \(<\) operator that will add the set elements from \( a(i) \) into the set \( i \)

Note

It would also be possible to run this example by explicitly defining the set \( i /i1, i2/; \) before the parameter declaration.

Attention

GAMS Transfer will overwrite all duplicate records when merging. The original values of \( a("i1") \) and \( a("i2") \) have been replaced with their new values when writing the Container in this example (see output below).

GAMS 38.1.0 Copyright (C) 1987-2022 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(16) 3 Mb
--- Initialize embedded library libembypcclib64.dylib
--- Execute embedded library libembypcclib64.dylib
--- test.gms(25) 3 Mb
--- Starting execution: elapsed 0:00:01.467
--- test.gms(19) 4 Mb
--- Initialize embedded library libembypcclib64.dylib
--- Execute embedded library libembypcclib64.dylib
--- i_0 value
  0 i1  1.0
  1 i2  2.0
  2 i3  3.0
  3 i4  4.0
  4 i5  5.0
  5 i6  6.0
  6 i7  7.0
  7 i8  8.0
  8 i9  9.0
*** Status: Normal completion
Example (Advanced Matrix Generation and Inversion w/ Write Operation)

```gams
set i / i1*i1000 /;
alias(i,j);

parameter a(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;

parameter inv_a(i,j);
parameter ident(i,j);

embeddedCode Python:
import gamstransfer as gt
import numpy as np
import time
gams.printLog("")
gams.printLog("")

s = time.time()
m = gt.Container(gams.db)
gams.printLog(f"read data: \{round(time.time() - s, 3)\} sec")

s = time.time()
A = m.data["a"].toDense()
gams.printLog(f"create matrix A: \{round(time.time() - s, 3)\} sec")

s = time.time()
invA = np.linalg.inv(A)
gams.printLog(f"calculate inv(A): \{round(time.time() - s, 3)\} sec")

s = time.time()
m.data["inv_a"].setRecords(invA)
gams.printLog(f"convert matrix to records for inv(A): \{round(time.time() - s, 3)\} sec")

s = time.time()
I = np.dot(A,invA)
tol = 1e-9
I[np.where((I<tol) & (I>-tol))] = 0
gams.printLog(f"calculate A*invA + small number cleanup: \{round(time.time() - s, 3)\} sec")

s = time.time()
m.data["ident"].setRecords(I)
gams.printLog(f"convert matrix to records for I: \{round(time.time() - s, 3)\} sec")

s = time.time()
m.write(gams.db, ["inv_a","ident"])
gams.printLog(f"write to GamsDatabase: \{round(time.time() - s, 3)\} sec")

gams.printLog("")
endEmbeddedCode inv_a, ident
display ident;
```

In this example we extend the example shown in Reading from GamsDatabase and GMD Objects to read data from GAMS, calculate a matrix inversion, do the matrix multiplication, and then write both the
\(A^{-1}\) and \(A^\top A^{-1}\) (i.e., the identity matrix) back to GAMS for display in the LST file. This data round trip highlights the benefits of using a GAMS Transfer Container (and the linked symbol structure) as the mechanism to move data – converting back and forth from a records format to a matrix format can be cumbersome, but here, GAMS Transfer takes care of all the indexing for the user.

The first few lines of GAMS code generates a 1000x1000 \(A\) matrix as a parameter (at execution time), we then define two more parameters that we will fill with results of the embedded Python code – specifically we want to fill a parameter with the matrix \(A^{-1}\) and we want to verify that another parameter (\(\text{ident}\)) contains the identity matrix (i.e., 1). Stepping through the code:

1. We start the embedded Python code section (execution time) by importing both GAMS Transfer and Numpy and by reading all the symbols that currently exist in the GamsDatabase. We must read in all this information in order to get the domain set information – GAMS Transfer needs these domain sets in order to generate matrices with the proper size.

2. Generate the matrix \(A\) by calling \(\text{.toDense()}\) on the symbol object in the Container.

3. Take the inverse of \(A\) with \(\text{np.linalg.inv()}\).

4. The Parameter symbol for \(\text{inv}_A\) already exists in the Container, but it does not have any records (i.e., \(\text{m.data["inv_a"].records is None}\) will evaluate to True). We use \(\text{.setRecords()}\) to convert the \(\text{invA}\) back into a records format.

5. We continue the computations by performing the matrix multiplication using \(\text{np.dot()}\) – we must clean up a lot of small numbers in 1.

6. The Parameter symbol for \(\text{ident}\) already exists in the Container, but it does not have any records. We use \(\text{.setRecords()}\) to convert 1 back into a records format.

7. Since we are calculating these parameter values at execution time, it is not possible to modify the domain set information (or even merge/replace it). Therefore we only want to write the parameter values to GAMS. We achieve this by writing a subset of the Container symbols out with the \(\text{m.write(gams.db, ["inv_a","ident"])}\) call. This partial write preserves symbol validity in the Container and it does not violate other GAMS requirements.

8. Finally, we can verify that the (albeit large) identity matrix exists in the LST file (or in another GDX file).

Note

It was not possible to just use \(\text{np.round}\) because small negative numbers that round to \(-0.0\) will be interpreted by GAMS Transfer as the GAMS EPS special value.

The output for this example is shown below:

```
GAMS 38.1.0  Copyright (C) 1987–2022 GAMS Development. All rights reserved
--- Starting compilation
--- matrix.gms(52)  3 Mb
--- Starting execution: elapsed 0:00:00.004
--- matrix.gms(11)  36 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---
--- read data: 1.083 sec
--- create matrix A:  0.016 sec
--- calculate inv(A):  0.032 sec
--- convert matrix to records for inv(A):  0.176 sec
```
7.5.2.24 ConstContainer (Rapid Read)

In the Create a Container section we describe how to use the main object class of GAMS Transfer – the Container. Many users of GAMS Transfer will rely on the Container for building their data pipeline, however some users will only be interested in post-processing data from a GAMS model run. This one-directional flow of data means that these users do not need some of the advanced Container features such as domain linking, matrix generation, domain checking, etc. The ConstContainer (i.e., a Constant Container) object class is a data-focused read-only object that will provide a snapshot of the data target being read – the ConstContainer can be created by reading a GDX file or a GamsDatabase/GMD object (an in memory representation of data used e.g. in embedded Python code).

7.5.2.24.1 Creating a ConstContainer

The ConstContainer shares many of the same methods and attributes that are in the Container class, which makes moving between the ConstContainer and the Container very simple. There are some important differences though:

1. The ConstContainer does not link any symbol data
2. The ConstContainer can only read from one source at a time – every new call of .read() will clear the data dict
3. The ConstContainer constructor will not read in any symbol records – this enables users to browse an unknown data source quickly (similar behavior to gdxdump).
4. The ConstContainer does not have a .write() method – a ConstContainer can be passed to the constructor of a Container which will enable data writing (however a copy of the data will be generated).
5. The user will never need to instantiate a symbol object and add it to the ConstContainer – the ConstContainer will internally generate its own set of (simplified) symbol classes and hold them in the .data attribute.

All of these differences were inspired by users that want to read the data as fast as possible and probe unknown data files without worrying about memory issues – ConstContainer provides users with a high level view of the data very quickly.

ConstContainer constructor

Creating a ConstContainer is a simple matter of initializing an object. For example:

```python
import gamstransfer as gt
h = gt.ConstContainer("out.gdx")
```

Note

This new ConstContainer object, here called h, will load all the symbol data from out.gdx but it will not load any of the records. To load records, users must use the .read() method.
### 7.5 Python API

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_from</td>
<td>str or GamsDatabase/GMD Object</td>
<td>Points to the source of the data being read into the ConstContainer</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>system_directory</td>
<td>str</td>
<td>Absolute path to GAMS system_directory</td>
<td>No</td>
<td>Attempts to find the GAMS installation by creating a GamsWorkspace object and loading the system_directory attribute.</td>
</tr>
</tbody>
</table>

The `ConstContainer` contains many of the same methods that are in the `Container` class, specifically:

**ConstContainer Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>describeAliases</td>
<td>create a summary table with descriptive statistics forAliases</td>
<td>symbols=None (None,str,list) - if None, assumes all aliases</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describeParameters</td>
<td>create a summary table with descriptive statistics forParameters</td>
<td>symbols=None (None,str,list) - if None, assumes all parameters</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describEquations</td>
<td>create a summary table with descriptive statistics forEquations</td>
<td>symbols=None (None,str,list) - if None, assumes all equations</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describeSets</td>
<td>create a summary table with descriptive statistics forSets</td>
<td>symbols=None (None,str,list) - if None, assumes all sets</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>describeVariables</td>
<td>create a summary table with descriptive statistics forVariables</td>
<td>symbols=None (None,str,list) - if None, assumes all variables</td>
<td>pandas.DataFrame</td>
</tr>
<tr>
<td>listAliases</td>
<td>list all aliases</td>
<td>-</td>
<td>list</td>
</tr>
<tr>
<td>listEquations</td>
<td>list all equations</td>
<td>types=None (list of equation types) - if None, assumes all types</td>
<td>list</td>
</tr>
<tr>
<td>listParameters</td>
<td>list all parameters</td>
<td>-</td>
<td>list</td>
</tr>
<tr>
<td>listSets</td>
<td>list all sets</td>
<td>-</td>
<td>list</td>
</tr>
<tr>
<td>listSymbols</td>
<td>list all symbols</td>
<td>-</td>
<td>list</td>
</tr>
<tr>
<td>listVariables</td>
<td>list all variables</td>
<td>types=None (list of variable types) - if None, assumes all types</td>
<td>list</td>
</tr>
<tr>
<td>read</td>
<td>main method to read load_from, can be provided with a list of symbols to read in subsets, records controls if symbol records are loaded or just metadata</td>
<td>load_from (str,GMD Object Handle,GamsDatabase Object) symbols=&quot;all&quot; (str, list) records=True (bool)</td>
<td>None</td>
</tr>
</tbody>
</table>
The structure of the DataFrames that are returned from the `describe` methods mirrors that in the `Container`; the user should reference Describing Data for detailed descriptions of the columns.

### 7.5.2.24.2 ConstContainer Symbol Objects

The `ConstContainer` uses a simplified symbol class structure to hold symbol specific information. The user will never need to directly instantiate these symbol classes (called `SimpleSet`, `SimpleParameter`, `SimpleVariable`, `SimpleEquation` and `SimpleAlias`); these symbol classes are nested under the `ConstContainer` class. This class structure is used to provide the feel of a read-only object.

While users do not need to instantiate any of the `Simple` symbol objects directly, they are available for users to probe. Many of the same `Container` symbol methods that generate summary statistics exist for the `ConstContainer` symbols. Specifically:

#### SimpleSet Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create <code>[*] * n</code> domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>is_singleton</td>
<td>bool if symbol is a singleton set</td>
<td>bool</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns <code>len(self.records)</code> if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to <code>domain_forwarding</code> state</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>

#### SimpleSet Methods

None
### SimpleParameter Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create &quot;[*] * n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>is_scalar</td>
<td>True if the len(self.domain) = 0</td>
<td>bool</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns len(self.records) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
</tbody>
</table>

### SimpleParameter Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues.EPS across all columns</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues.NA across all columns</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues.NEGINF across all columns</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues.POSINF across all columns</td>
<td>-</td>
<td>int or None</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues.UNDEF across all columns</td>
<td>-</td>
<td>int or None</td>
</tr>
</tbody>
</table>
### Application Programming Interfaces

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>findEps</td>
<td>find index positions of SpecialValues.EPS in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of SpecialValues.NA in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of SpecialValues.NEGINF in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of SpecialValues.POSINF in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of SpecialValues.Undef in value column</td>
<td>-</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>-</td>
<td>float or None</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the domain entry of records with a maximum value (return first instance only)</td>
<td>-</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the domain entry of records with a maximum absolute value (return first instance only)</td>
<td>-</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the domain entry of records with a minimum value (return first instance only)</td>
<td>-</td>
<td>list of str or None</td>
</tr>
</tbody>
</table>

### SimpleVariable Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;*&quot;] * n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
</tbody>
</table>
### Property Description Type Special Setter Behavior

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns <code>len(self.records)</code> if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to <code>domain_forwarding</code> state</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str type of variable</td>
<td>str</td>
<td>-</td>
</tr>
</tbody>
</table>

**SimpleVariable Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of <code>SpecialValues.EPS</code> across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>int or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>countNA</td>
<td>total number of <code>SpecialValues.NA</code> across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>int or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of <code>SpecialValues.NEGINF</code> across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>int or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of <code>SpecialValues.POSINF</code> across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>int or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of <code>SpecialValues.UNDEF</code> across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>int or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>findEps</td>
<td>find index positions of <code>SpecialValues.EPS</code> in column</td>
<td>column=&quot;level&quot;</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str)</td>
<td></td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of <code>SpecialValues.NA</code> in column</td>
<td>column=&quot;level&quot;</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str)</td>
<td></td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of <code>SpecialValues.NEGINF</code> in column</td>
<td>column=&quot;level&quot;</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str)</td>
<td></td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of <code>SpecialValues.POSINF</code> in column</td>
<td>column=&quot;level&quot;</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str)</td>
<td></td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of <code>SpecialValues.Undef</code> in column</td>
<td>column=&quot;level&quot;</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str)</td>
<td></td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns=&quot;level&quot;</td>
<td>float or None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(str.list)</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns=&quot;level&quot; (str, list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns=&quot;level&quot; (str, list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns=&quot;level&quot; (str, list)</td>
<td>float or None</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the domain entry of records with a maximum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the domain entry of records with a maximum absolute value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the domain entry of records with a minimum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
</tbody>
</table>

SimpleEquation Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol</td>
<td>int</td>
<td>setting is a shorthand notation to create [&quot;=&quot;] + n domains in symbol</td>
</tr>
<tr>
<td>domain_forwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>bool</td>
<td>no effect after records have been set</td>
</tr>
<tr>
<td>domain_labels</td>
<td>column headings for the records DataFrame</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_names</td>
<td>string version of domain names</td>
<td>list of str</td>
<td>-</td>
</tr>
<tr>
<td>domain_type</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>str</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>str</td>
<td>sets the GAMS name of the symbol</td>
</tr>
<tr>
<td>number_records</td>
<td>number of symbol records (i.e., returns len(self.records) if not None)</td>
<td>int</td>
<td>-</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>pandas.DataFrame</td>
<td>responsive to domain_forwarding state</td>
</tr>
<tr>
<td>summary</td>
<td>output a dict of only the metadata</td>
<td>dict</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>str type of variable</td>
<td>str</td>
<td>-</td>
</tr>
</tbody>
</table>
## SimpleEquation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues.EPS across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues.NA across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues.NEGINF across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues.POSINF across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>int or None</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues.UNDEF across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>int or None</td>
</tr>
<tr>
<td>findEps</td>
<td>find index positions of SpecialValues.EPS in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNA</td>
<td>find index positions of SpecialValues.NA in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findNegInf</td>
<td>find index positions of SpecialValues.NEGINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findPosInf</td>
<td>find index positions of SpecialValues.POSINF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>findUndef</td>
<td>find index positions of SpecialValues.UNDEF in column</td>
<td>column=&quot;level&quot; (str)</td>
<td>pandas.Index or None</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>float or None</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns=&quot;level&quot; (str.list)</td>
<td>float or None</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the domain entry of records with a maximum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the domain entry of records with a maximum absolute value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the domain entry of records with a minimum value (return first instance only)</td>
<td>column=&quot;level&quot; (str)</td>
<td>list of str or None</td>
</tr>
</tbody>
</table>
Example (reading only meta data w/ ConstContainer constructor)

```python
import gamstransfer as gt
h = gt.ConstContainer('trnsport.gdx')
In [1]: h.data
Out[1]:
{
    'i': <src.gamstransfer.ConstContainer.SimpleSet at 0x7fba484bf5b0>,
    'j': <src.gamstransfer.ConstContainer.SimpleSet at 0x7fba484bffd0>,
    'a': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba484bf880>,
    'b': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0f460>,
    'd': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0ff40>,
    'f': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0fa00>,
    'c': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0f160>,
    'x': <src.gamstransfer.ConstContainer.SimpleVariable at 0x7fba48b0f7c0>,
    'z': <src.gamstransfer.ConstContainer.SimpleVariable at 0x7fba48b0f3a0>,
    'cost': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0f7f0>,
    'supply': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0fd30>,
    'demand': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0fc70>
}
In [2]: h.describeParameters()
Out[2]:

<table>
<thead>
<tr>
<th>name</th>
<th>is_scalar</th>
<th>domain</th>
<th>type</th>
<th>dim</th>
<th>num</th>
<th>recs</th>
<th>sparsity</th>
<th>min_value</th>
<th>mean_value</th>
<th>max_value</th>
<th>where_min</th>
<th>where_max</th>
<th>count_pa</th>
<th>count_undef</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>False</td>
<td>[i]</td>
<td>regular</td>
<td>1</td>
<td>2</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>350.000</td>
<td>475.000</td>
<td>600.000</td>
<td>[seattle]</td>
<td>[san-diego]</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>False</td>
<td>[j]</td>
<td>regular</td>
<td>1</td>
<td>3</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>275.000</td>
<td>300.000</td>
<td>325.000</td>
<td>[topeka]</td>
<td>[new-york]</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>False</td>
<td>[i, j]</td>
<td>regular</td>
<td>2</td>
<td>6</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>0.126</td>
<td>0.176</td>
<td>0.225</td>
<td>[san-diego]</td>
<td>[topeka]</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>False</td>
<td>[i, j]</td>
<td>regular</td>
<td>2</td>
<td>6</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>1.400</td>
<td>1.950</td>
<td>2.500</td>
<td>[san-diego]</td>
<td>[topeka]</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>True</td>
<td>[]</td>
<td>none</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>90.000</td>
<td>90.000</td>
<td>90.000</td>
<td>None</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that in this example we make use of the convenience notation contained in the constructor to read in only the metadata of the `trnsport.gdx` file. This allows users to quickly explore the symbols contained in a file (or in-memory object) and it also explains why there are many None values in the columns of the `describeParameters()` method.

Example (reading all data w/ ConstContainer.read() method)

```python
import gamstransfer as gt
h = gt.ConstContainer()  
h.read("trnsport.gdx")
In [1]: h.data
Out[1]:
{
    'i': <src.gamstransfer.ConstContainer.SimpleSet at 0x7fba484bf5b0>,
    'j': <src.gamstransfer.ConstContainer.SimpleSet at 0x7fba484bffd0>,
    'a': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba484bf880>,
    'b': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0f460>,
    'd': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0ff40>,
    'f': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0fa00>,
    'c': <src.gamstransfer.ConstContainer.SimpleParameter at 0x7fba48b0f160>,
    'x': <src.gamstransfer.ConstContainer.SimpleVariable at 0x7fba48b0f7c0>,
    'z': <src.gamstransfer.ConstContainer.SimpleVariable at 0x7fba48b0f3a0>,
    'cost': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0f7f0>,
    'supply': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0fd30>,
    'demand': <src.gamstransfer.ConstContainer.SimpleEquation at 0x7fba48b0fc70>
}
In [2]: h.describeParameters()
Out[2]:

<table>
<thead>
<tr>
<th>name</th>
<th>is_scalar</th>
<th>domain</th>
<th>type</th>
<th>dim</th>
<th>num</th>
<th>recs</th>
<th>sparsity</th>
<th>min_value</th>
<th>mean_value</th>
<th>max_value</th>
<th>where_min</th>
<th>where_max</th>
<th>count_pa</th>
<th>count_undef</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>False</td>
<td>[i]</td>
<td>regular</td>
<td>1</td>
<td>2</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>350.000</td>
<td>475.000</td>
<td>600.000</td>
<td>[seattle]</td>
<td>[san-diego]</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>False</td>
<td>[j]</td>
<td>regular</td>
<td>1</td>
<td>3</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>275.000</td>
<td>300.000</td>
<td>325.000</td>
<td>[topeka]</td>
<td>[new-york]</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>False</td>
<td>[i, j]</td>
<td>regular</td>
<td>2</td>
<td>6</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>0.126</td>
<td>0.176</td>
<td>0.225</td>
<td>[san-diego]</td>
<td>[topeka]</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>False</td>
<td>[i, j]</td>
<td>regular</td>
<td>2</td>
<td>6</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>1.400</td>
<td>1.950</td>
<td>2.500</td>
<td>[san-diego]</td>
<td>[topeka]</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>True</td>
<td>[]</td>
<td>none</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>None</td>
<td>None</td>
<td>90.000</td>
<td>90.000</td>
<td>90.000</td>
<td>None</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>

In this example we make use of the `read()` method to retrieve both the metadata and the numerical records for all symbols in the GDX file – the `describeParameters()` method will now populate the DataFrame with additional summary statistics.
7.6 Matlab API

The Matlab API consists of the object-oriented GAMS API and GAMS Transfer Matlab.

7.6.1 GAMS API

GAMS Matlab API provides a Matlab programming interface to the General Algebraic Model System (GAMS). GAMS Matlab API objects allow a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (GAMSModelInstance).

Note

There also exist the data exchange tools GDXMRW and GAMS Transfer Matlab to exchange data between Matlab and GDX files. Note that GDXMRW and GAMS Transfer Matlab are likely to be more efficient in case of exchanging very large data sets. However, the GAMS Matlab API offers more features, for example, to start and modify a GAMS job.

The underlying GAMS engine relies to some extent on file based communication (e.g. the listing file) and other unmanaged resources. Since the GAMS Matlab API is based on the GAMS Java API, the use of external resources in the Java environment requires special attention. Hence, some objects need to be properly disposed before the Java garbage collector does its job.

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN or execute $load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base object GAMSWorkspace manages the anchor to the file system.

With the exception of GAMSWorkspace the objects in the gams package cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the GAMSModelInstance class).

Note

If you use multiple instances of the GAMSWorkspace in parallel, you should avoid using the same WorkingDirectory. Otherwise you may end up with conflicting file names.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GAMSModelInstances. Some solvers will not work in a multi-threaded application using GAMSModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GAMSModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

This version of the GAMS Matlab API also does not provide support for the following GAMS components: acronyms, GAMS compilation/execution errors, structured access to listing file, and solver options.

7.6.2 GAMS Transfer Matlab

7.7 R API

The R API consists of GAMS Transfer R.
7.7.1 GAMS Transfer R

GAMS Transfer is a package to maintain GAMS data outside a GAMS script in a programming language like Python, Matlab, or R. It allows the user to add GAMS symbols (Sets, Aliases, Parameters, Variables and Equations), to manipulate GAMS symbols, as well as read/write symbols to different data endpoints. GAMS Transfer's main focus is the highly efficient transfer of data between GAMS and the target programming language, while keeping those operations as simple as possible for the user. In order to achieve this, symbol records - the actual and potentially large-scale data sets - are stored in native data structures of the corresponding programming languages. The benefits of this approach are threefold: (1) The user is usually very familiar with these data structures, (2) these data structures come with a large tool box for various data operations, and (3) optimized methods for reading from and writing to GAMS can transfer the data as a bulk - resulting in the high performance of this package.

7.7.1.1 Getting Started

7.7.1.1.1 Install  The user must download and install the latest version of GAMS in order to install GAMS Transfer R. GAMS Transfer R can then be installed from either the source package or from the binary package.

The installation from the source will install `gamstransfer` and all of its dependencies.

```r
install.packages("[PathToGAMS]/apifiles/R/gamstransfer/source/gamstransfer_r.tar.gz", dependencies=TRUE)
```

The users can also install `gamstransfer` without compiling the package source code using `gamstransfer` binary package. The binary packages are platform dependent and the instructions for each supported platform are shown below.

Windows

```r
install.packages("[PathToGAMS]/apifiles/R/binary/gamstransfer.zip", type="binary")
```

Linux

```r
install.packages("[PathToGAMS]/apifiles/R/binary/gamstransfer.tar.gz")
```

Mac OS X

```r
install.packages("[PathToGAMS]/apifiles/R/binary/gamstransfer.tgz", type="binary")
```

GAMS Transfer R depends on packages `R6`, `assertthat`, `R.utils`, and `Rcpp`.

7.7.1.1.2 Examples

7.7.1.1.2.1 GDX Read  Reading in all symbols can be accomplished with one line of code (we reference data from the `trnsport.gms` example).

```r
m = Container$new("trnsport.gdx")
```

All symbol data is organized in the `data` field (a list) – `m$data[<symbol_name>]+$records` – records are stored as data frames.
7.7 R API

7.7.1.1.2.2 GDX Write

There are five symbol classes within GAMS Transfer R: Sets, Parameters, Variables, Equations, and Aliases. For purposes of this quick start, we show how to recreate the distance data structure from the `trnsport.gms` model (the parameter `d`). This brief example shows how users can achieve "GAMS-like" functionality, but within an R environment – GAMS Transfer R leverages the object oriented programming to simplify the syntax.

```r
m = Container$new()
# create the sets i, j
i = Set$new(m, "i", records=c("seattle", "san-diego"), description = "supply")
j = Set$new(m, "j", records=c("new-york", "chicago", "topeka"), description = "markets")
# add "d" parameter -- domain linked to set objects i and j
d = Parameter$new(m, "d", c(1, j), description = "distance in thousands of miles")
# create some data as a generic data frame
dist = data.frame(
  from = c("seattle", "seattle", "seattle", "san-diego", "san-diego", "san-diego"),
  to = c("new-york", "chicago", "topeka", "new-york", "chicago", "topeka"),
  thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
# setRecords will automatically convert the dist data frame into
# a standard data frame format
d$setRecords(dist)
# write the GDX
m$write("out.gdx")
```

This example shows a few fundamental features of GAMS Transfer R:

1. A Container is analogous to a GDX file
2. Symbols will always be linked to a Container (notice that we always pass the Container reference `m` to the symbol constructor)
3. Records can be added to a symbol with the `setRecords()` method, through the `records` constructor argument (internally calls `setRecords()`), or through directly setting the `records` field. GAMS Transfer R will convert many common R data structures into a standard format.
4. Domain linking is possible by passing domain set objects to other symbols – this will also enable domain checking (violations will show up as `<NA>` as shown in this example)
5. Writing a GDX file can be accomplished in one line with the `write()` method.

7.7.1.1.2.3 Full Example

It is possible to use GAMS Transfer R to recreate the `trnsport.gms` results in GDX form. As part of this example, we also introduce the Container methods `write()` method (and generate `new.gdx`). We will discuss it in more detail in the following section: Data Exchange with GDX.

```r
# create an empty Container object
m = Container$new()
# add sets
i = Set$new(m, "i", records=c("seattle", "san-diego"), description = "supply")
j = Set$new(m, "j", records=c("new-york", "chicago", "topeka"), description = "markets")
# add parameters
a = Parameter$new(m, "a", c("*")), description="capacity of plant i in cases")
b = Parameter$new(m, "b", j, description="demand at market j in cases")
d = Parameter$new(m, "d", c(1, j), description="distance in thousands of miles")
f = Parameter$new(m, "f", records=90, description="freight in dollars per case per thousand miles")
c = Parameter$new(m, "c", c(1, j), description="transport cost in thousands of dollars per case")
# set parameter records
cap = data.frame(plant = c("seattle", "san-diego"), n_cases = c(350, 600))
a$setRecords(cap)
dem = data.frame(market = c("new-york", "chicago", "topeka"), n_cases = c(325, 300, 275))
b$setRecords(dem)
dist = data.frame(
  from = c("seattle", "seattle", "seattle", "san-diego", "san-diego", "san-diego"),
  to = c("new-york", "chicago", "topeka", "new-york", "chicago", "topeka"),
  thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
d$setRecords(dist)
# (i,j) = f * d(i,j) / 1000;
cost = d$records
cost$value = f$records$value * cost$value/1000
```
c$setRecords(cost)
# add variables
q = data.frame(
from = c("seattle", "seattle", "seattle",
"san-diego", "san-diego", "san-diego"),
to = c("new-york", "chicago", "topeka",
"new-york", "chicago", "topeka"),
level = c(50, 300, 0, 275, 0, 275),
marginal = c(0, 0, 0.036, 0, 0.009, 0)
)
x = Variable$new(m, "x", "positive", c(i, j), records=q, description="shipment quantities in cases")

z = Variable$new(m, "z", records=data.frame(level = 153.675),
description="total transportation costs in thousands of dollars")

# add equations

cost = Equation$new(m, "cost", "eq", description="define objective function")
supply = Equation$new(m, "supply", "leq", i, description="observe supply limit at plant i")
demand = Equation$new(m, "demand", "geq", j, description="satisfy demand at market j")

# set equation records


cost$setRecords(data.frame(level = 0, marginal = 1, lower = 0, upper = 0))
supplies = data.frame(
from = c("seattle", "san-diego"),
level = c(350, 550),
marginal = c(SpecialValues$EPS, 0),
lower = c(SpecialValues$NEGINF, SpecialValues$NEGINF),
upper = c(350, 600)
)
supply$setRecords(supplies)
demands = data.frame(
from = c("new-york", "chicago", "topeka"),
level = c(325, 300, 275),
marginal = c(0.225, 0.153, 0.126),
lower = c(325, 300, 275)
)
demand$setRecords(demands)

m$write("new.gdx")

It can be observed from the above example that a typical work flow for writing using GAMS Transfer R is creating a container, filling it with symbols (Sets, Parameters, Variables, Equations, and Aliases), and write it to a GDX file. To read a GDX file, a Container can simply be initialized with the GDX file name as an argument.

These examples introduced the reader to the GAMS Transfer R syntax. In the remaining sections, we will present details about the core functionality and dig further into the syntax.

### 7.7.1.2 Manual

#### 7.7.1.2.1 Container

Storing, manipulating, and transforming sparse data requires that it lives within an environment – this data can then be linked together to enable various operations. In GAMS Transfer R, we refer to this “environment” as the Container, it is the main repository for storing and linking our data. Linking data enables data operations like implicit set growth, domain checking, data format transformations (to dense/sparse matrix formats), etc. A Container also enables different read/write operations.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadFrom</td>
<td>string</td>
<td>Name of the GDX file being read into the Container</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>systemDirectory</td>
<td>string</td>
<td>Absolute path to GAMS system directory</td>
<td>No</td>
<td>Attempts to find the GAMS installation by checking path system variable and looking for the GAMS executable.</td>
</tr>
</tbody>
</table>

Creating a Container is a simple matter of initializing an object. For example:

m = Container$new()
This new `Container` object, here called `m`, contains a number of convenient fields and methods that allow the user to interact with the symbols that are in the `Container`. Some of these methods are simply used to filter out different types of symbols, other methods are used to numerically characterize the data within each symbol.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Special Setter Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>main list that is used to store all symbol data</td>
<td>list</td>
<td>-</td>
</tr>
</tbody>
</table>

### 7.7.1.2.1.1 Container fields
Symbols are organized in the `Container` under the `data` field. The `$` notation (`m$data`) is used to access the underlying named list; symbols in this named list can then be retrieved with the standard bracket (or double bracket) notation (`m$data[<symbol.name>]`).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>addAlias</td>
<td>add an Alias</td>
<td>name (string) aliasWith (Set, Alias)</td>
<td>Alias object</td>
</tr>
<tr>
<td>addEquation</td>
<td>add an Equation</td>
<td>name (string) type (string) domain=NULL (string, list) records=NULL (data frame, vector, NULL) domainForwarding=FALSE (logical) description=&quot;&quot; (string)</td>
<td>Equation object</td>
</tr>
<tr>
<td>addParameter</td>
<td>add a Parameter</td>
<td>name (string) domain=NULL (string, list) records=NULL (data frame, array, matrix) domainForwarding=FALSE (logical) description=&quot;&quot; (string)</td>
<td>Parameter object</td>
</tr>
<tr>
<td>addSet</td>
<td>add a Set</td>
<td>name (string) domain=&quot;*&quot; (string, list) isSingleton=FALSE (logical) records=NULL (data frame, array, matrix) domainForwarding=FALSE (logical) description=&quot;&quot; (string)</td>
<td>Set object</td>
</tr>
<tr>
<td>addVariable</td>
<td>add a Variable</td>
<td>name (string) type=&quot;free&quot; (string) domain=NULL (string, list) records=NULL (data frame, array, matrix) domainForwarding=FALSE (logical) description=&quot;&quot; (string)</td>
<td>Variable object</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>describeAliases</td>
<td>create a summary table with descriptive statistics for Aliases</td>
<td>symbols=NULL (string,list) - if NULL, assumes all Aliases</td>
<td>data frame</td>
</tr>
<tr>
<td>describeParameters</td>
<td>create a summary table with descriptive statistics for Parameters</td>
<td>symbols=NULL (string,list) - if NULL, assumes all parameters</td>
<td>data frame</td>
</tr>
<tr>
<td>describeEquations</td>
<td>create a summary table with descriptive statistics for Equations</td>
<td>symbols=NULL (string,list) - if NULL, assumes all equations</td>
<td>data frame</td>
</tr>
<tr>
<td>describeSets</td>
<td>create a summary table with descriptive statistics for Sets</td>
<td>symbols=NULL (string,list) - if NULL, assumes all sets</td>
<td>data frame</td>
</tr>
<tr>
<td>describeVariables</td>
<td>create a summary table with descriptive statistics for Variables</td>
<td>symbols=NULL (string,list) - if NULL, assumes all variables</td>
<td>data frame</td>
</tr>
<tr>
<td>getSymbols</td>
<td>returns a list of object references for symbols</td>
<td>symbols (character, vector, list)</td>
<td>list</td>
</tr>
<tr>
<td>getUniverseSet</td>
<td>provides a universe for all symbols</td>
<td>-</td>
<td>list</td>
</tr>
<tr>
<td>isValid</td>
<td>TRUE if all symbols in the Container are valid</td>
<td>verbose=FALSE (logical) force=FALSE (logical)</td>
<td>logical</td>
</tr>
<tr>
<td>listAliases</td>
<td>list all aliases (is Valid=NULL), list all valid aliases (is Valid=TRUE), list all invalid aliases (is Valid=FALSE) in the container</td>
<td>isValid=NULL (logical)</td>
<td>vector</td>
</tr>
<tr>
<td>listEquations</td>
<td>list all equations (is Valid=NULL), list all valid equations (is Valid=TRUE), list all invalid equations (is Valid=FALSE) in the container</td>
<td>isValid=NULL (logical) types=NULL (list or vector of equation types) - if NULL, assumes all types</td>
<td>vector</td>
</tr>
<tr>
<td>listParameters</td>
<td>list all parameters (is Valid=NULL), list all valid parameters (is Valid=TRUE), list all invalid parameters (is Valid=FALSE) in the container</td>
<td>isValid=NULL (logical)</td>
<td>vector</td>
</tr>
<tr>
<td>listSets</td>
<td>list all sets (is Valid=NULL), list all valid sets (is Valid=TRUE), list all invalid sets (is Valid=FALSE) in the container</td>
<td>isValid=NULL (logical)</td>
<td>vector</td>
</tr>
</tbody>
</table>
### 7.7 R API

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>listSymbols</td>
<td>list all symbols (isValid=NULL), list all valid symbols (isValid=TRUE), list invalid symbols (isValid=FALSE) in the container</td>
<td>isValid=NULL (logical)</td>
<td>vector</td>
</tr>
<tr>
<td>listVariables</td>
<td>list all variables (isValid=NULL), list all valid variables (isValid=TRUE), list invalid variables (isValid=FALSE) in the container</td>
<td>isValid=NULL (logical) types=NULL (list or vector of variable types) - if NULL, assumes all types</td>
<td>vector</td>
</tr>
<tr>
<td>read</td>
<td>main method to read loadFrom, can be provided with a list of symbols to read in subsets, records controls if symbol records are loaded or just metadata</td>
<td>loadFrom (string) symbols=&quot;all&quot; (string, list) records=TRUE (logical)</td>
<td>NULL</td>
</tr>
<tr>
<td>removeSymbols</td>
<td>symbols to remove from the Container, also sets the symbols' refContainer to NULL.</td>
<td>symbols (string.list)</td>
<td>NULL</td>
</tr>
<tr>
<td>renameSymbol</td>
<td>rename a symbol in the Container</td>
<td>oldName (string), newName (string)</td>
<td>NULL</td>
</tr>
<tr>
<td>reorderSymbols</td>
<td>reorder symbols in order to avoid domain violations</td>
<td>-</td>
<td>NULL</td>
</tr>
<tr>
<td>write</td>
<td>main bulk write method to a writeTo target</td>
<td>writeTo (string) compress=FALSE (logical) uelPriority=NULL (string.list)</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### 7.7.1.2.2 Container methods

#### 7.7.1.2.2 Symbols

In GAMS Transfer R, a symbol is either a GAMS Set, Parameter, Variable, Equation, Alias. In GAMS Transfer R, a symbol cannot live on its own, but is always part of a container.

Create a symbol

There are two different ways to create a GAMS set and add it to a Container.

1. Use symbol constructor for Set, Parameter, Variable, Equation, Alias

   ```
   m = Container$new()
p = m$addParameter("p")
   ```

1. Use the Container methods addSet, addParameter, addVariable, addEquation, addAlias (which internally calls the symbol constructor)

   ```
   m = Container$new()
p = Parameter$newIn("p")
   ```
Add Symbol Records

Three possibilities exist to assign symbol records:

1. Setting the argument records in the symbol constructor/Container method (internally calls setRecords)
2. Using the symbol method setRecords
3. Setting the field records directly

If the data is in a convenient format, a user may want to pass the records directly within the set constructor. This is an optional keyword argument and internally the symbol constructor will simply call the setRecords method. The symbol method setRecords is a convenience method that transforms the given data into an approved data frame format (see Standard Data Formats). Many native R data types can be easily transformed into data frames, so the setRecords method will accept a number of different types for input. The setRecords method is called internally on any data structure that is passed through the records argument.

The examples of setting records using the above-mentioned methods can be found in the respective documentation for each symbol ( Set, Parameter, Variable, Equation, and Alias ).

### 7.7.1.2.2.1 Set

#### Set Constructor

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>String, List of domains given either as a string (&quot;*&quot; for universe set) or as a reference to a Set object</td>
<td>No</td>
<td>&quot;*&quot;</td>
</tr>
<tr>
<td>isSingleton</td>
<td>logical</td>
<td>Indicates if set is a singleton set (TRUE) or not (FALSE)</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>records</td>
<td>string vector, data frame</td>
<td>Symbol records</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>logical</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

#### Set Fields
### Field Description Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>string</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing &quot;*&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>logical</td>
</tr>
<tr>
<td>domainLabels</td>
<td>column headings for the records data frame</td>
<td>list of string</td>
</tr>
<tr>
<td>domainNames</td>
<td>string version of domain names</td>
<td>list of string</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>string</td>
</tr>
<tr>
<td>isSingleton</td>
<td>logical if symbol is a singleton set</td>
<td>logical</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>string</td>
</tr>
<tr>
<td>numberRecords</td>
<td>number of symbol records (i.e., returns nrow(self$records) if not NULL)</td>
<td>integer</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>data frame</td>
</tr>
<tr>
<td>refContainer</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
</tr>
<tr>
<td>summary</td>
<td>output a list of only the metadata</td>
<td>list</td>
</tr>
</tbody>
</table>

### Set Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>vector</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE</td>
<td>verbose=FALSE force=FALSE</td>
<td>logical</td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard data frame formatted records</td>
<td>records (string vector, list, data frame)</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Adding Set Records

Three possibilities exist to assign symbol records to a set: We show a few examples of ways to create differently structured sets.

Example #1 - Create a 1D set from a vector

```r
n = Container$new()
> i = Set$new(n, "i", records = c("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
# i = Set$new(n, "i")
# $i$setRecords(c("seattle", "san-diego"))
# $n$setSet("i", records= c("seattle", "san-diego"))
```
Example #2 - Create a 2D set from a list

```r
n = Container$new()
k = Set$new(n, "k", c("*", "*"), records=list("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
# k = Set$new(n, "k", c("*", "*"))
# $setRecords(list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# $addSet("k", c("*", "*"), records=list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# k = $addSet("k", c("*", "*"), records=list("seattle", "san-diego"))
# $setRecords(list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# $addSet("k", c("*", "*"))
# NOTE: the above syntax is also equivalent to -
# k = $addSet("k", c("*", "*"))
# $data["k"]$setRecords(list("seattle", "san-diego"))
> k
  # records
  uni
  1 element
  1 seattle
  2 san-diego
```

Example #3 - Create a 1D set from a data frame slice

```r
m = Container$new()
dist = data.frame(
  from = c("seattle", "seattle", "seattle", "san-diego", "san-diego", "san-diego"),
  to = c("new-york", "chicago", "topeka", "new-york", "chicago", "topeka"),
  thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
l = Set$new(m, "l", records = unique(dist["from"]))
# NOTE: the above syntax is equivalent to -
# l = Set$new(m, "l")
# $setRecords(unique(dist["from"]))
# NOTE: the above syntax is also equivalent to -
# $addSet("l", records=unique(dist["from"]))
# NOTE: the above syntax is also equivalent to -
# l = $addSet("l")
# $setRecords(unique(dist["from"]))
# NOTE: the above syntax is also equivalent to -
# $addSet("l")
# $data["l"]$setRecords(unique(dist["from"]))
> l
  # records
  uni
  1 element
  1 seattle
  2 san-diego
```

Set element text is very handy when labeling specific set elements within a set. A user can add a set element text directly with a set element. Note that it is not required to label all set elements, as can be seen in the following example.

Example #4 - Add set element text

```r
m = Container$new()
i = Set$new(m, "i", records = data.frame(c("seattle", "san-diego", "washington_d.c"),
  c("home of sub pop records", "", "former gams hq")))
# NOTE: the above syntax is equivalent to -
# i = Set$new(m, "i")
# $setRecords(c("seattle", "san-diego", "washington_d.c"),
# c("home of sub pop records", "", "former gams hq"))
# $setRecords(i_rec)
```
### Example #5 - Directly set records (1D set)

```r
n = Container$new()
i = Set$new(m, "i", description = "supply")
# create a standard format data frame
df_i = data.frame(uni_1 = c("seattle", "san-diego"),
                  element_text = c("", ""))
# need to create categorical column type, referencing elements already in df_i
df_i$uni_1 = factor(df_i$uni_1, ordered = TRUE)
# set the records directly
i$records = df_i
> i$isValid()
[1] TRUE
```

Stepping through this example we take the following steps:

1. Create an empty Container
2. Create a GAMS set `i` in the Container, but do not set the `records`
3. Create a data frame (manually, in this example) taking care to follow the standard format
4. The data frame has the right shape and column labels so we can proceed to set the records.
5. We need to cast the `uni_1` column as a factor, so we create a custom ordered category type using `factor`
6. Finally, we set the records directly by passing a reference to `df_i` into the symbol `records` attribute.

The setter function of `records` checks that a data frame is being set, but does not check validity. Thus, as a final step, we call the `$isValid()` method to verify that the symbol is valid.

### Note

Users can debug their data frames by running `<symbol_name>$isValid(verbosë=TRUE)` to get feedback about their data.
Example #6 - Directly set records (1D subset)

```
n = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"), description="supply")
j = Set$new(m, "j", i, description="supply")
# create a standard format data frame
df.j = data.frame(i = c("seattle"), "element_text" = c("")
# create the categorical column type
df.j$i = factor(df.j$i, levels = i$records[, 1], ordered = TRUE)
# set the records
j$records = df.j
> jisValid()
[1] TRUE
```

This example is more subtle in that we want to create a set `j` that is a subset of `i`. We create the set `i` using the `setRecords` method but then set the records directly for `j`. There are two important details to note: 1) the column labels in `df.j` now reflect the standard format for a symbol with a domain set (as opposed to the universe) and 2) we create the factors by referencing the parent set (`i`) for the `levels` (instead of referencing itself).

### 7.7.1.2 Parameter

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the <code>Container</code> object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>domain</td>
<td>list</td>
<td>List of domains given either as a string (&quot;*&quot; for universe set) or as a reference to a Set object, an empty domain list will create a scalar parameter</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>logical</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

Parameter Constructor

Parameter Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>string</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing &quot;*&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>logical</td>
</tr>
<tr>
<td>domainLabels</td>
<td>column headings for the records data frame</td>
<td>list of string</td>
</tr>
<tr>
<td>domainNames</td>
<td>string version of domain names</td>
<td>list of string</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>string</td>
</tr>
<tr>
<td>isScalar</td>
<td>TRUE if the length(self$domain) = 0</td>
<td>logical</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>string</td>
</tr>
<tr>
<td>numberRecords</td>
<td>number of symbol records (i.e., returns nrow(self$records) if not NULL)</td>
<td>integer</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>data frame</td>
</tr>
<tr>
<td>refContainer</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
</tr>
<tr>
<td>shape</td>
<td>a vector describing the array dimensions if records were converted with $toDense()</td>
<td>vector</td>
</tr>
<tr>
<td>summary</td>
<td>output a list of only the metadata</td>
<td>list</td>
</tr>
</tbody>
</table>

Parameter Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues$EPS in value column</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues[&quot;NA&quot;] in value column</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues$NEGINF in value column</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues$POSINF in value column</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues$UNDEF in value column</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>vector</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value in value column</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value in value column</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value in value column</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value in value column</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE</td>
<td>verbose=FALSE force=FALSE</td>
<td>logical</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard data frame records</td>
<td>records (many types)</td>
<td>NULL</td>
</tr>
<tr>
<td>toDense</td>
<td>convert symbol to a dense matrix or array format</td>
<td>column = value</td>
<td>array or matrix</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the row number in records data frame with a maximum value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the row number in records data frame with a maximum absolute value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the row number in records data frame with a minimum value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
</tbody>
</table>

Adding Parameter Records

Three possibilities exist to assign symbol records to a parameter: We show a few examples of ways to create differently structured parameters:

Example #1 - Create a GAMS scalar

```r
n = Container$new()
pic = Parameter$new(n, "pi", records = 3.14159)
# NOTE: the above syntax is equivalent to -
# pi = Parameter$new(n, "pi")
# pi$setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# $addParameter("pi", records=3.14159)
# NOTE: the above syntax is also equivalent to -
# $addParameter("pi")
# pi$setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# $addParameter("pi")
# $data["pi"]$setRecords(3.14159)
pic$records
value
1 3.14159
```

Note

GAMS Transfer R will still convert scalar values to a standard format (i.e., a data frame with a single row and column).

Example #2 - Create a 2D parameter (defined over a set) from a data frame slice

```r
dist = data.frame(
  from = c("seattle", "seattle", "seattle", "san-diego", "san-diego", "san-diego"),
  to = c("new-york", "chicago", "topeka", "new-york", "chicago", "topeka"),
  thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
```

m = Container$new()
i = Set$new(m, "i", "+", records = unique(dist$from))

j = Set$new(m, "j", "+", records = unique(dist$to))
a = Parameter$new(m, "a", c(i, j), records = dist)

> a$toDense()

[,1] [,2] [,3]
[1,] 2.5 1.7 1.8
[2,] 2.5 1.8 1.4

# use a$toDense() to create a new (and identical) parameter a2
a2 = Parameter$new(m, "a2", c(i, j), records = a$toDense())

> a2$records

 i  j 2 value
1  seattle new-york 2.5
2  seattle chicago 1.7
3  seattle topeka 1.8
4  san-diego new-york 2.5
5  san-diego chicago 1.8
6  san-diego topeka 1.4

Example #3 - Create a 2D parameter from an array using setRecords

m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:5))

j = Set$new(m, "j", records=paste0("j", 1:5))

# create the parameter with linked domains (these will control the
# shape of the symbol)
a = Parameter$new(m, "a", c(i, j))

# here we use the shape field to easily generate a dense random array
a$records(array(runif(prod(a$shape()), min = 1, max = 10),
            dim = a$shape())

> a$toDense()

[1,] 3.837345 3.632743 9.003275 4.097475 8.608477
[2,] 7.217257 2.465452 3.286330 2.366017 8.822535
[3,] 8.421044 8.546226 5.403918 2.286660 6.319740
[4,] 3.960100 8.538932 2.210829 2.437113 5.324722
[5,] 1.333846 4.508688 7.411279 5.653044 7.248775

As with Sets, the primary advantage of the setRecords method is that GAMS Transfer will convert many
different (and convenient) data types into the standard data format (data frame). Users that require
higher performance will want to directly pass the Container a reference to a valid data frame, thereby
skipping some of these computational steps. This places more burden on the user to pass the data in a
valid standard form, but it speeds the records setting process. In this section, we walk the user through
an example of how to set records directly.

Example #4 - Correctly set records (directly)

df = data.frame(h=1:8760, m=1:60, s=1:60)
df$value = runif(nrow(df), min = 0, max = 100)

m = Container$new()
hrs = Set$new(m, "h", records=unique(df$h))
mins = Set$new(m, "m", records=unique(df$m))
secs = Set$new(m, "s", records=unique(df$s))

df$h = factor(df$h, ordered = TRUE)
df$m = factor(df$m, ordered = TRUE)
df$s = factor(df$s, ordered = TRUE)

a = Parameter$new(m, "a", c(hrs, mins, secs))

# set records
a$records = df

> a$isInvalid()

[1] TRUE

In this example, we create a large parameter (31,536,000 records and 8880 unique domain elements – we
mimic data that is labeled for every second in one year) and assign it to a parameter with a$records.
GAMS Transfer R requires that all domain columns must be ordered factors. The records setter function
does very little work other than checking if the object being set is a data frame. This places more
responsibility on the user to create a data frame that complies with the standard format. In Example #1
we take care to properly reference the factor from the domain sets – and in the end a$isInvalid() = TRUE.

Users will need to use the isValid(verbos=TRUE) method to debug any structural issues. As an
example, we incorrectly generate categorical data types by passing the data frame constructor the generic
factor argument. This creates factor columns, but they are not ordered and they do not reference the
underlying domain set. These errors result in a being invalid.
Example #5 - Incorrectly set records (directly)

```r
df = data.frame(h1 = paste0("h", 1:8760), m2 = paste0("m", 1:60), s3 = paste0("s", 1:60))
df$value = runif(nrow(df), min = 0, max = 100)
hrs = Set$new(m, "h", records = unique(df$h1))
mins = Set$new(m, "m", records = unique(df$m2))
secs = Set$new(m, "s", records = unique(df$s3))
df$h1 = factor(df$h1)
df$m2 = factor(df$m2)
df$s3 = factor(df$s3)
a = Parameter$new(m, "a", c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] FALSE
> a$isValid(verbose=TRUE)
Domain information in column h1 must be an ORDERED factor
[1] FALSE
```

7.7.1.2.2.3 Variable

Variable Constructor

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>Type of variable being created [binary, integer, positive, negative, free, sos1, sos2, semicont, semiint]</td>
<td>No</td>
<td>free</td>
</tr>
<tr>
<td>domain</td>
<td>list, vector</td>
<td>List, vector of domains given either as a string (&quot;*&quot; for universe set) or as a reference to a Set object, an empty domain list will create a scalar variable</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>logical</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

Variable Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>string</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing &quot;*&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>logical</td>
</tr>
<tr>
<td>domainLabels</td>
<td>column headings for the records data frame</td>
<td>list of string</td>
</tr>
</tbody>
</table>
### Field Description Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>domainNames</td>
<td>string version of domain names</td>
<td>list of string</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>string</td>
</tr>
<tr>
<td>numberRecords</td>
<td>number of symbol records (i.e., returns nrow(self$records) if not NULL)</td>
<td>integer</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>data frame</td>
</tr>
<tr>
<td>refContainer</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
</tr>
<tr>
<td>shape</td>
<td>a vector describing the array dimensions if records were converted with $toDense()</td>
<td>vector</td>
</tr>
<tr>
<td>summary</td>
<td>output a list of only the metadata</td>
<td>list</td>
</tr>
<tr>
<td>type</td>
<td>string type of variable</td>
<td>list</td>
</tr>
</tbody>
</table>

### Variable Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues$EPS across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues[&quot;NA&quot;] across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues$NEGINF across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of SpecialValues$POSINF across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of SpecialValues$UNDEF across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>vector</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if Force=TRUE</td>
<td>verbose=FALSE, force=FALSE</td>
<td>logical</td>
</tr>
</tbody>
</table>
### Method | Description | Arguments/Defaults | Returns
--- | --- | --- | ---
| `setRecords` | main convenience method to set standard data frame records | `records` (many types) | `NULL` |
| `toDense` | convert symbol to a dense matrix or array format | column = value | array or matrix |
| `whereMax` | find the row number in records data frame with a maximum value (return the first instance only) | - | integer |
| `whereMaxAbs` | find the row number in records data frame with a maximum absolute value (return the first instance only) | - | integer |
| `whereMin` | find the row number in records data frame with a minimum value (return the first instance only) | - | integer |

### Adding Variable Records

Three possibilities exist to assign symbol records to a variable: We show a few examples of ways to create differently structured variables:

**Example #1 - Create a GAMS scalar variable**

```r
m = Container$new()
p = Variable$m('pi', records = data.frame(level = 3.14159))
# NOTE: the above syntax is equivalent to -
# p = Variable$m('pi', 'free')
# p$setRecords(data.frame(level = 3.14159))
# NOTE: the above syntax is also equivalent to -
# m$addVariable('pi', 'free', records=data.frame(level = 3.14159))
> p$records
     level marginal lower upper scale
1 3.14159       0   -Inf   Inf   1
```

**Example #2 - Create a 2D positive variable, specifying no numerical data**

```r
m = Container$new()
v = Variable$m('v', 'positive', c('seattle', 'san-diego'), c('chicago', 'madison'))
> v$records
     uni_1 uni_2 level marginal lower upper scale
1  seattle   san-diego     0       0     0   -Inf  Inf   1
2  chicago    madison     0       0     0   -Inf  Inf   1
```

**Example #3 - Create a 2D variable (defined over a set) from a matrix**

```r
m = Container$new()
set1 = Set$m('i', '*', records = paste0('i', 1:5))
set2 = Set$m('j', '*', records = paste0('j', 1:5))
# creating records for parameter a
ij = list(set1 = paste0('i', 1:5), set2 = paste0('j', 1:5))
df = rev(expand.grid(rev(ij)))
da = Parameter$m('a', c(set1, set2), records = df)
# create a free variable and set the level and marginal attributes from matricies
v = Variable$m('v', 'domain = c(i, j), records = list(level = a$toDense(), marginal = a$toDense())
# if not specified, the toDense() method will convert the level values to a matrix`
As with Sets, the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (data frame). Users that require higher performance will want to directly pass the `Container` a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process. In this section, we walk the user through an example of how to set records directly.

Example #4 - Correctly set records (directly)

```R
df = data.frame(h1 = paste0("h", 1:8760), m2 = paste0("m", 1:60),
                 s3 = paste0("s", 1:60))
df$level = runif(nrow(df), min = 0, max = 100)
df$marginal = 0.0
df$lower = SpecialValues$NEGINF
df$upper = SpecialValues$POSINF
df$scale = 1.0
m = Container$new()
hrs = Set$new(m, "h", records = unique(df$h1))
nins = Set$new(m, "m", records = unique(df$m2))
secs = Set$new(m, "s", records = unique(df$s3))
dfh1 = factor(dfh1, ordered = TRUE)
dfh2 = factor(dfh2, ordered = TRUE)
dfh3 = factor(dfh3, ordered = TRUE)
# set records
a$records = df
> a$isValid()
[1] TRUE
```

In this example, we create a large variable (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to a variable with `a$records`. GAMS Transfer R requires that all domain columns must be ordered factors. The `records` setter function does very little work other than checking if the object being set is a data frame. This places more responsibility on the user to create a data frame that complies with the standard format. In Example #1 we take care to properly reference the factor from the domain sets – and in the end `a$isValid()` = `TRUE`.

As with Set and Parameters, users can use the `$isValid(verboselTRUE)` method to debug any structural issues.

### 7.7.1.2.2.4 Equation

#### Equation Constructor

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the <code>Container</code> object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>Type of equation being created [eq (or E/e), geq (or G/g), leq (or L/l), nonbinding (or N/n), external (or X/x)]</td>
<td>No</td>
<td>free</td>
</tr>
</tbody>
</table>
### Application Programming Interfaces

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>list, vector</td>
<td>List, vector of domains given either as a string (“*” for universe set) or as a reference to a Set object, an empty domain list will create a scalar equation</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>records</td>
<td>many</td>
<td>Symbol records</td>
<td>No</td>
<td>NULL</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>logical</td>
<td>Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Description of symbol</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

#### Equation Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>string</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing &quot;*&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>logical</td>
</tr>
<tr>
<td>domainLabels</td>
<td>column headings for the records data frame</td>
<td>list of string</td>
</tr>
<tr>
<td>domainNames</td>
<td>string version of domain names</td>
<td>list of string</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>string</td>
</tr>
<tr>
<td>numberRecords</td>
<td>number of symbol records (i.e., returns nrow(self$records) if not NULL)</td>
<td>integer</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>data frame</td>
</tr>
<tr>
<td>refContainer</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
</tr>
<tr>
<td>shape</td>
<td>a vector describing the array dimensions if records were converted with $toDense()</td>
<td>vector</td>
</tr>
<tr>
<td>summary</td>
<td>output a list of only the metadata</td>
<td>list</td>
</tr>
<tr>
<td>type</td>
<td>string type of equation</td>
<td>list</td>
</tr>
</tbody>
</table>

#### Equation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>countEps</td>
<td>total number of SpecialValues$Epsilon across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countNA</td>
<td>total number of SpecialValues[[&quot;NA&quot;]] across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countNegInf</td>
<td>total number of SpecialValues$NEGINF across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Arguments/Defaults</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>---------</td>
</tr>
<tr>
<td>countPosInf</td>
<td>total number of <code>SpecialValues$POSINF</code> across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>countUndef</td>
<td>total number of <code>SpecialValues$UNDEF</code> across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>findDomainViolations</td>
<td>get the index of records that contain any domain violations</td>
<td>-</td>
<td>vector</td>
</tr>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxValue</td>
<td>get the maximum value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMinValue</td>
<td>get the minimum value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMeanValue</td>
<td>get the mean value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>getMaxAbsValue</td>
<td>get the maximum absolute value across all columns</td>
<td>columns = &quot;level&quot;</td>
<td>numeric</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE</td>
<td>verbose=FALSE, force=FALSE</td>
<td>logical</td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard data frame records</td>
<td>records (many types)</td>
<td>NULL</td>
</tr>
<tr>
<td>toDense</td>
<td>convert symbol to a dense matrix or array format</td>
<td>column = value</td>
<td>array or matrix</td>
</tr>
<tr>
<td>whereMax</td>
<td>find the row number in records data frame with a maximum value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>whereMaxAbs</td>
<td>find the row number in records data frame with a maximum absolute value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>whereMin</td>
<td>find the row number in records data frame with a minimum value (return the first instance only)</td>
<td>-</td>
<td>integer</td>
</tr>
</tbody>
</table>

Adding Equation Records

Three possibilities exist to assign symbol records to an equation: We show a few examples of ways to create differently structured equations:
Example #1 - Create a GAMS scalar equation

```gams
m = Container
pi = Equation
new(m, "pi", records = data.frame(level = 3.14159))
# NOTE: the above syntax is equivalent to -
# pi = Equation
$ new(m, "pi", "free")
# pi
$ setRecords(data.frame(level = 3.14159))
> pi$records
  level marginal lower upper scale
  1  3.14159 0  -Inf Inf  1
```

Example #2 - Create a 2D positive equation, specifying no numerical data

```gams
m = Container
e = Equation
new(m, "e", "eq", c("seattle", "chicago"), (("san-diego", c("madison"))))
> e
$ records
  uni
  1 uni
  2 level marginal lower upper scale
  1 seattle san-diego 0  0  -Inf Inf  1
  2 chicago    madison 0  0  -Inf Inf  1
```

Example #3 - Create a 2D equation (defined over a set) from a matrix

```gams
m = Container
i = Set
new(m, "i", ",", records = paste0("i", 1:5))
j = Set
new(m, "j", ",", records = paste0("j", 1:5))
# creating records for parameter a
ij = list(i = paste0("i", 1:5), j = paste0("j", 1:5))
df = rev(expand.grid(rev(ij)))
df$value = 1:25
a = Parameter
new(m, "a", c(i, j), records = df)
# create a free variable and set the level and marginal attributes from matrices
e = Equation
new(m, "e", "nonbinding", domain = c(i, j), records = list(level = a$toDense(), marginal = a$toDense()))
> e$records
  i j level marginal lower upper scale
  1 1    1  1 -Inf Inf  1
  2 1    2  6 -Inf Inf  1
  3 1    3 11 -Inf Inf  1
  4 1    4 16 -Inf Inf  1
  5 1    5 21 -Inf Inf  1
  6 2    2  2 -Inf Inf  1
  7 2    7  7 -Inf Inf  1
  8 2    12 12 -Inf Inf  1
  9 2    17 17 -Inf Inf  1
 10 2    22 22 -Inf Inf  1
 11 3    3  3 -Inf Inf  1
 12 3    8  8 -Inf Inf  1
 13 3    13 13 -Inf Inf  1
 14 3    18 18 -Inf Inf  1
 15 3    23 23 -Inf Inf  1
 16 4    4  4 -Inf Inf  1
 17 4    9  9 -Inf Inf  1
 18 4    14 14 -Inf Inf  1
 19 4    19 19 -Inf Inf  1
 20 4    24 24 -Inf Inf  1
 21 5    5  5 -Inf Inf  1
 22 5    10 10 -Inf Inf  1
 23 5    15 15 -Inf Inf  1
 24 5    20 20 -Inf Inf  1
 25 5    25 25 -Inf Inf  1
```

As with sets, parameters, and variables the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (data frame). Users that require higher performance will want to directly pass the `Container` a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process. In this section, we walk the user through an example of how to set records directly.
Example #4 - Correctly set records (directly)

```r
df = data.frame(h = paste0("h", 1:8760), m = paste0("m", 1:60),
                s = paste0("s", 1:60))
df$level = runif(nrow(df), min = 0, max = 100)
df$marginal = 0.0
df$lower = SpecialValues$NEGINF
df$upper = SpecialValues$POSINF
df$scale = 1.0
m = Container$new()
n = Container$new()
hrs = Set$new(m, "h", records = unique(df$h))
nins = Set$new(m, "n", records = unique(df$m))
secs = Set$new(m, "s", records = unique(df$s))
dfh,l = factor(df$h, ordered = TRUE)
dfm,s_2 = factor(df$m, ordered = TRUE)
dfs,s_3 = factor(df$s, ordered = TRUE)
a = Equation$new(m, "a", "eq", domain = c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] TRUE
```

In this example, we create a large equation (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to an equation with `a$records`. GAMS Transfer R requires that all domain columns must be ordered factors. The `records` setter function does very little work other than checking if the object being set is a data frame. This places more responsibility on the user to create a data frame that complies with the standard format. In Example #1 we take care to properly reference the factor from the domain sets – and in the end `a$isValid() = TRUE`.

As with sets, parameters, and variables, users can use the `$isValid(verbos=TRUE)` method to debug any structural issues.

### 7.7.1.2.2.5 Alias

**Alias Constructor**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Container</td>
<td>A reference to the Container object that the symbol is being added to</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name of symbol</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>aliasWith</td>
<td>Set object</td>
<td>set object to create an alias for</td>
<td>Yes</td>
<td>-</td>
</tr>
</tbody>
</table>

Example - Creating an alias from a set

GAMS Transfer R only stores the reference to the parent set as part of the alias structure – most properties that are called from an alias object simply point to the properties of the parent set (with the exception of `refContainer`, `name`, and `aliasWith`). It is possible to create an alias from another alias object. In this case, a recursive search will be performed to find the root parent set – this is the set that will ultimately be stored as the `aliasWith` field. We can see this behavior in the following example:

```r
n = Container$new()
i = Set$new(m, "i", records = paste0("i", 1:5))
ip = Alias$new(m, "ip", i)
ipp = Alias$new(m, "ipp", ip)
> ipp$aliasWith$name
[1] "ip"
> ipp$aliasWith$name
[1] "i"
```
Alias Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>description of symbol</td>
<td>string</td>
</tr>
<tr>
<td>dimension</td>
<td>dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing &quot;∗&quot;</td>
<td>integer</td>
</tr>
<tr>
<td>domainForwarding</td>
<td>flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth)</td>
<td>logical</td>
</tr>
<tr>
<td>domainLabels</td>
<td>column headings for the records data frame</td>
<td>list of string</td>
</tr>
<tr>
<td>domainNames</td>
<td>string version of domain names</td>
<td>list of string</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on state of domain links</td>
<td>string</td>
</tr>
<tr>
<td>isSingleton</td>
<td>logical if symbol is a singleton set</td>
<td>logical</td>
</tr>
<tr>
<td>name</td>
<td>name of symbol</td>
<td>string</td>
</tr>
<tr>
<td>numberRecords</td>
<td>number of symbol records (i.e., returns nrow(self$records) if not NULL)</td>
<td>integer</td>
</tr>
<tr>
<td>records</td>
<td>the main symbol records</td>
<td>data frame</td>
</tr>
<tr>
<td>refContainer</td>
<td>reference to the Container that the symbol belongs to</td>
<td>Container</td>
</tr>
<tr>
<td>summary</td>
<td>output a list of only the metadata</td>
<td>list</td>
</tr>
</tbody>
</table>

Alias Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Arguments/Defaults</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCardinality</td>
<td>get the full cartesian product of the domain</td>
<td>-</td>
<td>integer</td>
</tr>
<tr>
<td>getSparsity</td>
<td>get the sparsity of the symbol w.r.t the cardinality</td>
<td>-</td>
<td>numeric</td>
</tr>
<tr>
<td>isValid</td>
<td>checks if the symbol is in a valid format, throw exceptions if</td>
<td>verbose=FALSE, force=FALSE</td>
<td>logical</td>
</tr>
<tr>
<td></td>
<td>verbose=TRUE, recheck a symbol if force=TRUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setRecords</td>
<td>main convenience method to set standard data frame formatted records</td>
<td>records (string vector, list, data frame)</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Adding Alias Records

The linked structure of Aliases offers some unique opportunities to access some of the setter functionality of the parent set. Specifically, GAMS Transfer allows the user to change the domain, description, dimension, and records of the underlying parent set as a shorthand notation. We can see this behavior if we look at a modified Example #1 from R_GAMSTRANSFER_ADD_SET_RECORDS.
Example - Creating set records through an alias link

```r
m = Container$new()
i = Set$new(m, "i")
i$p$description = "adding new descriptive set text"
i$p$domain = c("*", "*")
i = list(paste0("i", 1:3), paste0("j", 1:3))
i$p$setRecords(rev(expand.grid(rev(i))))

> i$p$domain
[[1]]
[1] "*"
[[2]]
[1] "*

> i$p$records
1 1 1 1
2 1 1 1
3 1 1 1
4 1 1 1
5 1 1 1
6 1 1 1
7 1 1 1
8 1 1 1
9 1 1 1
```

Note

An alias `isValid()` = TRUE when the underlying parent set is also valid – if the parent set is removed from the Container the alias will no longer be valid.

7.7.1.3 Additional Features

7.7.1.3.1 Validating Data

GAMS Transfer R requires that the records for all symbols exist in a standard format (Standard Data Formats) in order for them to be understood and written successfully. It is certain possible that the data could end up in a state that is inconsistent with the standard format (especially if setting symbol attributes directly). GAMS Transfer R includes the `isValid()` method in order to determine if a symbol is valid and ready for writing; this method returns a logical. For example, we create two valid sets and then check them with `isValid()` to be sure.

Example (valid data)

```r
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego", "washington dc"))
j = Set$new(m, "j", i, records = c("san-diego", "washington dc"))

> i$p$isValid()
[1] TRUE
> j$p$isValid()
[1] TRUE
> m$p$isValid()
[1] TRUE
```

Now we create some data that is invalid due to domain violations in the set j.

```r
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego", "washington dc")))
j = Set$new(m, "j", i, records = c("grayslake", "washington dc"))

> i$p$isValid()
[1] TRUE
> j$p$records
1 1 1 1
2 <NA>
> j$p$isValid()
```

Note

It is possible to run `isValid()` on both the Container as well as the symbol object – `isValid()` will also return a logical if there are any invalid symbols in the Container object.
In this example, we know that the validity of the data is compromised by the domain violations, but there could be other subtle discrepancies that must be remedied before writing data. The user can get more detailed error reporting if the `verbose` argument is set to `TRUE`. For example:

```r
> isValid(verbose=TRUE)
Symbol 'records' contain domain violations; ensure that all domain elements have been mapped properly to a factor
```

The `$isValid()` method checks:

1. If the symbol belongs to a `Container`
2. If all domain set symbols exist in the Container
3. If all domain set symbols objects are valid
4. If records are a data frame (or `None`)
5. The shape of the records is congruent with the dimensionality of the symbol
6. If records column headings are in standard format
7. If all domain columns are `factors` and also ordered
8. If all domain factors are referenced properly (`records` for referenced domain sets cannot be `NULL` in order to create categoricals properly)
9. If there are any domain violations
10. If there are any duplicate domain members
11. That all data columns are type `numerical`
12. To make sure that all domain categories are type `string`

Note

Calling `$isValid()` too often may have a significant impact on the performance.

### 7.7.1.3.2 Domain Forwarding

GAMS includes the ability to define sets directly from data using the implicit set notation (see: Implicit Set Definition (or: Domain Defining Symbol Declarations)). This notation has an analogue in GAMS Transfer R called `domainForwarding`.

**Note**

It is possible to recursively update a subset tree in GAMS Transfer R.

Domain forwarding is available as an argument to all symbol object constructors; the user would simply need to pass `domainForwarding=TRUE`.

In this example, we have raw data that is in the `dist` data frame, and we want to send the domain information into the `i` and `j` sets – we take care to pass the set objects as the domain for parameter `c`.

```r
m = Container$new()
i = Set$new(m, "i")
j = Set$new(m, "j")
dist = data.frame(
    from = c("seattle", "seattle", "seattle", "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka", "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
```
c = Parameter$new(x, 'c', c(i, j), records = dist, domainForwarding = TRUE)
> i$records
  uni_1 element_text
1   seattle
4   san-diego
> j$records
  uni_1 element_text
1   new-york
2   chicago
3    topeka
> c
  records
1    i    j2 value
1   seattle new-york 2.5
2   seattle chicago 1.7
3   seattle    topeka 1.8
4   san-diego new-york 2.5
5   san-diego chicago 1.8
6   san-diego    topeka 1.4

Note

The element order in the sets i and j mirrors that in the raw data.

In this example, we show that domain forwarding will also work recursively to update the entire set lineage – the domain forwarding occurs at the creation of every symbol object. The correct order of elements in set i is (z, a, b, c) because the records from j are forwarded first, and then the records from k are propagated through (back to i).

m = Container$new()
i = Set$new(m, 'i')
j = Set$new(m, 'j', i, records = c('z'), domainForwarding = TRUE)
k = Set$new(m, 'k', j, records = c('a', 'b', 'c'), domainForwarding = TRUE)
> i$records
  uni_1 element_text
1    z
2   a
3   b
4   c
> j$records
  i1 element_text
1    z
2   a
3   b
4   c
> k$records
  j1 element_text
1   a
2   b
3   c

7.7.1.3.3 Describing Data

The methods describeSets, describeParameters, describeVariables, and describeEquations allow the user to get a summary view of key data statistics. The returned data frame aggregates the output for a number of other methods (depending on symbol type). A description of each Container method is provided in the following subsections:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list,string</td>
<td>A list of sets in the Container to include in the output.</td>
<td>No</td>
<td>NULL (if NULL specified, will assume all sets)</td>
</tr>
</tbody>
</table>

7.7.1.3.3.1 describeSets

Returns: data frame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
</tbody>
</table>
### Field / Statistic Description

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAlias</td>
<td>logical if the symbol is an Alias</td>
</tr>
<tr>
<td>isSingleton</td>
<td>logical if the set is a singleton set</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>numRecs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - numberRecs/cardinality</td>
</tr>
</tbody>
</table>

#### Example #1

```r
m = Container$new("transport.gdx")
> m$describeSets()
name isAlias isSingleton domain domainType dim numRecs cardinality
1 i FALSE FALSE * none 1 2 NA
2 j FALSE FALSE * none 1 3 NA
sparsity
1 NA
2 NA
```

#### Argument Type Description Required Default

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, string</td>
<td>A list of parameters in the Container to include in the output</td>
<td>No</td>
<td>NULL (if NULL specified, will assume all parameters)</td>
</tr>
</tbody>
</table>

### 7.7.1.3.3.2 describeParameters

Returns: data frame

The following table includes a short description of the column headings in the return.

#### Field / Statistic Description

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>isScalar</td>
<td>logical if the symbol is a scalar (i.e., dimension = 0)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>numRecs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>minValue</td>
<td>min value in data</td>
</tr>
<tr>
<td>meanValue</td>
<td>mean value in data</td>
</tr>
<tr>
<td>maxValue</td>
<td>max value in data</td>
</tr>
<tr>
<td>whereMin</td>
<td>row number min value (if multiple, returns only first occurrence)</td>
</tr>
<tr>
<td>whereMax</td>
<td>row number of max value (if multiple, returns only first occurrence)</td>
</tr>
<tr>
<td>countEps</td>
<td>number of SpecialValues$\text{EPS}$ in data</td>
</tr>
<tr>
<td>countNa</td>
<td>number of SpecialValues[&quot;NA&quot;] in data</td>
</tr>
<tr>
<td>countUndef</td>
<td>number of SpecialValues$\text{UNDEF}$ in data</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - numRecs/cardinality</td>
</tr>
</tbody>
</table>

```r
m = Container$new("transport.gdx")
> m$describeParameters()
name isScalar domain domainType dim numRecs minValue meanValue maxValue
1 a FALSE i regular 1 2 350.000 475.0000 600.0000
```
2 b FALSE j regular 1 3 275.000 300.0000 325.000
5 c FALSE 1 j regular 2 6 0.126 0.1755 0.225
3 d FALSE 1 j regular 2 6 1.400 1.9500 2.500
4 f TRUE NA none 0 1 90.000 90.0000 90.000

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, string</td>
<td>A list of variables in the <a href="#">Container</a> to include in the output</td>
<td>No</td>
<td>NULL (if NULL specified, will assume all variables)</td>
</tr>
</tbody>
</table>

### 7.7.1.3.3.3 describeVariables

Returns: data frame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>type</td>
<td>type of variable (i.e., binary, integer, positive, negative, free, sos1, sos2, semicont, semiint)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>numRecs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - numRecs/cardinality</td>
</tr>
<tr>
<td>minLevel</td>
<td>min value in the level</td>
</tr>
<tr>
<td>meanLevel</td>
<td>mean value in the level</td>
</tr>
<tr>
<td>maxLevel</td>
<td>max value in the level</td>
</tr>
<tr>
<td>whereMaxAbsLevel</td>
<td>row number of max(abs(level)) in data</td>
</tr>
<tr>
<td>countEpsLevel</td>
<td>number of SpecialValues$$EPS in level</td>
</tr>
<tr>
<td>minMarginal</td>
<td>min value in the marginal</td>
</tr>
<tr>
<td>meanMarginal</td>
<td>mean value in the marginal</td>
</tr>
<tr>
<td>maxMarginal</td>
<td>max value in the marginal</td>
</tr>
<tr>
<td>whereMaxAbsMarginal</td>
<td>row number of max(abs(marginal)) in data</td>
</tr>
<tr>
<td>countEpsMarginal</td>
<td>number of SpecialValues$$EPS in marginal</td>
</tr>
</tbody>
</table>

```r
m = Container$\$nev("transport.gdx")
> m$describeVariables()
nname type domain domainType dim numRecs cardinality sparsity minLevel
1 x positive 1 j regular 2 6 6 0 0.000
2 z free NA none 0 1 NA NA 153.675
meanLevel maxLevel whereMaxAbsLevel countEpsLevel minMarginal meanMarginal
1 150.000 300.0000 2 0 0 0.0075
2 153.675 153.675 1 0 0 0.0000
maxMarginal whereMaxAbsMarginal countEpsMarginal
1 0.036 3 1
2 0.000 1 1
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, string</td>
<td>A list of equations in the <a href="#">Container</a> to include in the output</td>
<td>No</td>
<td>NULL (if NULL specified, will assume all variables)</td>
</tr>
</tbody>
</table>
7.7.1.3.3.4 describeEquations  Returns: data frame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>type</td>
<td>type of variable (i.e., binary, integer, positive, negative, free, sos1, sos2, semiint)</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>numRecs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>1 - numRecs/cardinality</td>
</tr>
<tr>
<td>minLevel</td>
<td>min value in the level</td>
</tr>
<tr>
<td>meanLevel</td>
<td>mean value in the level</td>
</tr>
<tr>
<td>maxLevel</td>
<td>max value in the level</td>
</tr>
<tr>
<td>whereMaxAbsLevel</td>
<td>row number of max(abs(level)) in data</td>
</tr>
<tr>
<td>countEpsLevel</td>
<td>number of SpecialValues$\epsilon$ in level</td>
</tr>
<tr>
<td>minMarginal</td>
<td>min value in the marginal</td>
</tr>
<tr>
<td>meanMarginal</td>
<td>mean value in the marginal</td>
</tr>
<tr>
<td>maxMarginal</td>
<td>max value in the marginal</td>
</tr>
<tr>
<td>whereMaxAbsMarginal</td>
<td>row number of max(abs(marginal)) in data</td>
</tr>
<tr>
<td>countEpsMarginal</td>
<td>number of SpecialValues$\epsilon$ in marginal</td>
</tr>
</tbody>
</table>

n = Container$\$new("transp.gdx")
> n$describeEquations()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbols</td>
<td>list, string</td>
<td>A list of aliases in the Container to include in the output.</td>
<td>No</td>
<td>NULL (if NULL specified, will assume all aliases)</td>
</tr>
</tbody>
</table>

7.7.1.3.3.5 describeAliases  Returns: data frame

The following table includes a short description of the column headings in the return.

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the symbol</td>
</tr>
<tr>
<td>aliasWith</td>
<td>name of the parent set</td>
</tr>
<tr>
<td>isSingleton</td>
<td>logical if an alias of a singleton set</td>
</tr>
<tr>
<td>domain</td>
<td>domain labels for the symbol</td>
</tr>
<tr>
<td>domainType</td>
<td>none, relaxed or regular depending on the symbol state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field / Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dim</td>
<td>dimension</td>
</tr>
<tr>
<td>numberRecs</td>
<td>number of records in the symbol</td>
</tr>
<tr>
<td>cardinality</td>
<td>cartesian product of the domain information</td>
</tr>
<tr>
<td>sparsity</td>
<td>$1 - \text{numberRecs}/\text{cardinality}$</td>
</tr>
</tbody>
</table>

Example #1

```r
m = Container$new()
i = Set$new(m, "i", records = paste0("i",1:5))
j = Set$new(m, "j", records = paste0("j",1:10))
ip = Alias$new(m, "ip", i)
ipp = Alias$new(m, "ipp", ip)
jp = Alias$new(m, "jp", j)
> m$describeAliases()
name aliasWith isSingleton domain domainType dim numberRecs cardinality
1 ip i FALSE none 1 5 NA
2 ipp i FALSE none 1 5 NA
3 jp j FALSE none 1 10 NA
```

7.7.1.3.4 Matrix Generation GAMS Transfer R stores data in a "flat" format, that is, one record entry per data frame row. However, it is often necessary to convert this data format into a matrix/array format – GAMS Transfer R enables users to do this with relative ease using the `toDense` symbol methods. This method will return a dense N-dimensional array (matrix for 2-Dimensions) with each dimension corresponding to the GAMS symbol dimension; it is possible to output an array up to 20 dimensions (a GAMS limit).

Example (1D data w/o domain linking (i.e., a relaxed domain))

```r
m = Container$new()
a = Parameter$new(m, "a", "i", records = data.frame(c("a","c"), c(1,3)))
> i$records
 uni_1 element_text
 1 a
 2 b
 3 c
 4 d
> a$toDense()
[1] 1 0 3
```

Note that the parameter `a` is not linked to another symbol, so when converting to a matrix, the indexing is referenced to the data structure in `a$records`. Defining a sparse parameter `a` over a set `i` allows us to extract information from the `i` domain and construct a very different dense matrix, as the following example shows:

```r
m = Container$new()
i = Set$new(m, "i", records = c("a", "b", "c", "d"))
a = Parameter$new(m, "a", i, records = data.frame(c("a","c"), c(1,3)))
> a$toDense()
[1] 1 0 3 0
```

Example (2D data w/ domain linking)

```r
m = Container$new()
i = Set$new(m, "i", records = c("a", "b", "c", "d"))
a = Parameter$new(m, "a", c(1, 1), records =
data.frame(c("a","c"), c("a","c"), c(1,3)))
> i$records
 uni_1 element_text
 1 a
```

Example (2D data w/ domain linking)
The Universe Set

A Unique Element List (UEL) (aka the "universe" or "universal set") is an \((i,s)\) pair where \(i\) is an identification number for a string \(s\). GAMS uses UELs to efficiently store domain entries of a record by storing the UEL ID \(i\) of a domain entry instead of the actual string \(s\). This avoids storing the same string multiple times. The concept of UELs also exists in R and is called a "factor". GAMS Transfer R leverages these types in order to efficiently store strings and enable domain checking within the R environment.

Each domain column in a data frame can be a factor, the effect is that each symbol maintains its own list of UELs per dimension. R lets the user choose if levels in a factor should be regarded as ordered. GAMS Transfer R relies exclusively on ordered factors (in order for a symbol to be valid it must have only ordered factors). By using ordered factors, GAMS Transfer R will order the UEL such that elements appear in the order in which they appeared in the data (which is how GAMS defines the UEL). **GAMSTransfer** allows the user to reorder the UELs with the `uelPriority` argument in the `$write()` method.

GAMS Transfer R does not keep track of the UEL separately from other symbols in the Container, it will be created internal to the `$write()` method and is based on the order in which data is added to the container. The user can access the current state of the UEL with the `$getUniverseSet()` container method. For example, we set a two dimensional set:

```r
m = Container$new()
j = Set$new(m, "j", c("*", "*"), records = data.frame(c("i1","i2"), c("j1","j2")))
```

R also allows the user to modify (rename, reorder, etc.) factor levels. These methods may be useful for advanced users, but most users will probably find that modifying the original data structures and resetting the symbol records provides a simpler solution. The design of GAMS Transfer R should enable the user to quickly move data back and forth, without worrying about the deeper mechanics of factors.

Removing Symbols

Removing symbols from a Container is easy when using the `removeSymbols` container method; this method accepts either a string or a list of string.

Attention

Once a symbol has been removed, it is possible to have hanging references as domain links in other symbols. The user will need to repair these other symbols with the proper domain links in order to avoid validity errors.

Reordering Symbols

In order to write the contents of the Container, it is required that the symbols are sorted such that, for example, a Set used as a domain of another symbol appears before that symbol. The Container will try to establish a valid ordering when writing the data. This type of situation could be encountered if the user is adding and removing many symbols (and perhaps rewriting symbols with the same name) – users should attempt to only add symbols to a Container once, and care must be taken when naming. The method `reorderSymbols` attempts to fix symbol ordering problems. The following example shows how this can occur:
Example Symbol reordering

```r
m = Container$new()
i = Set$new(m, "i", records = paste0("i", 1:5))
j = Set$new(m, "j", i, records = paste0("i", 1:3))
> names(m$data)
[1] "i" "j"
# now we remove the set i and recreate the data
m$removeSymbols("i")
i = Set$new(m, "i", records = paste0("i", 1:5))
> m$isValid()
[1] FALSE
Since the link to i is broken, Set j is now invalid. The user has to manually set the domain again.
# fix the domain reference in the set j
j$domain = i
> names(m$data)
[1] "j" "i"
Now even though j is valid, the symbols are out of order. The order can be fixed as follows.
# calling reorderSymbols() will order the list properly,
# but the domain reference in j is now broken
m$reorderSymbols()
> names(m$data)
[1] "i" "j"
> m$isValid()
[1] TRUE
```

7.7.1.3.8 GAMS Special Values The GAMS system contains five special values: UNDEF (undefined), NA (not available), EPS (epsilon), +INF (positive infinity), -INF (negative infinity). These special values must be mapped to their R equivalents. GAMS Transfer R follows the following convention to generate the 1:1 mapping:

- +INF is mapped to Inf
- -INF is mapped to -Inf
- EPS is mapped to -0.0 (mathematically identical to zero)
- NA is mapped to a NA
- UNDEF is mapped to NaN

GAMS Transfer R syntax is designed to quickly get data into a form that is usable in further analyses or visualization. The user does not need to remember these constants as they are provided within the class `SpecialValues` as `SpecialValues$POSINF`, `SpecialValues$NEGINF`, `SpecialValues$EPS`, `SpecialValues$"NA"`, and `SpecialValues$UNDEF`. Some examples are shown below.

```r
n = Container$new()
x = Parameter$new(
m, "x", c("*"),
records = data.frame(paste0("i", 1:6), c(1, SpecialValues$"POSINF", SpecialValues$"NEGINF", SpecialValues$"EPS", SpecialValues$"NA"),
SpecialValues$"UNDEF")),
description = "special values"
)
The following data frame for x would look like:
> x$records
  uni_1 value
  1   11   1
  2   12   Inf
  3   13   -Inf
  4   14   0
  5   15   NA
  6   16   NaN
Note
The syntax `SpecialValues$NA` is not allowed in R. Therefore, to access NA, one has to use `SpecialValues$"NA"`. As shown in the example above, double bracket syntax works for other special values too.
Standard Data Formats

This section is meant to introduce the standard format that GAMS Transfer R expects for symbol records. It has already been mentioned that we store data as a data frame, but there is an assumed structure to the column headings and column types that will be important to understand. GAMS Transfer R includes convenience functions in order to ease the burden of converting data from a user-centric format to one that is understood by GAMS Transfer R. However, advanced users will want to convert their data first and add it directly to the Container.

Set Records Standard Format

All set records (including singleton sets) are stored as a data frame with \( n \) number of columns, where \( n \) is the dimensionality of the symbol + 1. The first \( n-1 \) columns include the domain elements while the last column includes the set element explanatory text. Records are organized such that there is one record per row.

The names of the domain columns follow a pattern of \(<\text{domain}\_\text{set}\_\text{name}>_<\text{index}\_\text{position}>\); a symbol dimension that is referenced to the universe is labeled \( \text{uni}_<\text{index}\_\text{position}> \). The explanatory text column is called \text{element}\_\text{text} and must take the last position in the data frame.

All domain columns must be \text{factors} and the \text{element}\_\text{text} column must be a \text{string} type.

Some examples:

```r
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
j = Set$new(m, "j", c(i, "*"), records=data.frame(c("seattle", "san-diego"), c("new-york", "st-louis")))
k = Set$new(m, "k", 1, isSingleton=TRUE, records=c("seattle"))

# i$records
  uni_1 element\_text
1 seattle
2 san-diego
# j$records
  i 1 uni_2 element\_text
1 seattle new-york
2 san-diego st-louis
# k$records
  i 1 element\_text
1 seattle
```

Parameter Records Standard Format

All parameter records (including scalars) are stored as a data frame with \( n \) number of columns, where \( n \) is the dimensionality of the symbol + 1. The first \( n-1 \) columns include the domain elements while the last column includes the numerical value of the records. Records are organized such that there is one record per row. Scalar parameters have zero dimension, therefore they only have one column and one row.

The names of the domain columns follow a pattern of \(<\text{domain}\_\text{set}\_\text{name}>_<\text{index}\_\text{position}>\); a symbol dimension that is referenced to the universe is labeled \( \text{uni}_<\text{index}\_\text{position}> \). The value column is called \text{value} and must take the last position in the data frame.

All domain columns must be \text{factors} and the \text{value} column must be a \text{numeric} type. GAMS Transfer R requires that all the factor levels are of type \text{string}.

Some examples:

```r
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
a = Parameter$new(m, "a", "x", records=data.frame(c("seattle", "san-diego"), c(50, 100)))
b = Parameter$new(m, "b", c(i, "*"), records=data.frame(c("seattle", "san-diego"), c("new-york", "st-louis"), c(32.2, 123)))
c = Parameter$new(m, "c", records=90)

# a$records
```

Variable/Equation Records Standard Format

Variables and equations share the same standard data format. All records (including scalar variables/equations) are stored as a data frame with \( n \) number of columns, where \( n \) is the dimensionality of the symbol + 5. The first \( n-5 \) columns include the domain elements while the last five columns include the numerical values for different attributes of the records. Records are organized such that there is one record per row. Scalar variables/equations have zero dimension, therefore they have five columns and one row.

The names of the domain columns follow a pattern of \(< \text{domain set name} >, < \text{index position} >\); a symbol dimension that is referenced to the universe is labeled \( \text{uni} < \text{index position} > \). The attribute columns are called level, marginal, lower, upper, and scale. These attribute columns must appear in this order. Attributes that are not supplied by the user will be assigned the default GAMS values for that variable/equation type. It is possible to not pass any attributes, GAMS Transfer R would then simply assign default values to all attributes.

All domain columns must be factors and the attribute columns must be a numeric type. GAMS Transfer R requires that all the factor levels are of type string.

Some examples:

```r
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
a = Variable$new(m, "a", "free", domain=i, records=data.frame(city = c("seattle", "san-diego"), level = c(50, 100)))
```

```r
> a$records i level marginal lower upper scale
1 seattle 50 0 -Inf Inf 1
2 san-diego 100 0 -Inf Inf 1
```

7.7.1.3.10 Data Exchange with GDX

Up until now, we have been focused on using GAMS Transfer R to create symbols in an empty Container using the symbol constructors (or their corresponding container methods). These tools will enable users to ingest data from many different formats and add them to a Container – however, it is also possible to read in symbol data directly from GDX files using the read container method. In the following sections, we will discuss this method in detail as well as the write method, which allows users to write out to new GDX files.

7.7.1.3.10.1 Reading from GDX

There are two main ways to read in GDX based data.

- Pass the file path directly to the Container constructor (will read all symbols and records)
- Pass the file path directly to the read method (default read all symbols, but can read partial files)
The first option here is provided for convenience and will, internally, call the read method. This method will read in all symbols as well as their records. This is the easiest and fastest way to get data out of a GDX file and into your R environment. For the following examples, we leverage the GDX output generated from the transport.gms model file.

```R
m = Container$new("transport.gdx")
> names(m$data)
[1] "i" "j" "a" "b" "d" "f" "c" "x" "z" "cost" "supply" "demand"
> m.describeParameters()
name isScalar domain domainType dim numRecs minValue meanValue maxValue
1 a FALSE i regular 1 2 350.000 475.0000 600.000
2 b FALSE j regular 1 3 275.000 300.0000 325.000
5 c FALSE i j regular 2 6 0.126 0.1755 0.225
3 d FALSE i j regular 2 6 1.400 1.9500 2.500
4 f TRUE NA none 0 1 90.000 90.0000 90.000
whereMin whereMax countEps countNa countUndef cardinality sparsity
1 1 2 0 0 0 2 0
2 3 1 0 0 0 3 0
5 6 1 0 0 0 6 0
3 6 1 0 0 0 6 0
4 1 1 0 0 NA NA
```

A user could also read in data with the read method as shown in the following example.

**Example (reading full data w/ read method)**

```R
m = Container$new()
> m$read("transport.gdx")
> names(m$data)
[1] "i" "j" "a" "b" "d" "f" "c" "x" "z" "cost" "supply" "demand"
```

It is also possible to read in a partial GDX file with the read method, as shown in the following example:

```R
m = Container$new()
> m$read("transport.gdx", "x")
> names(m$data)
[1] "x"
> m$data$x$records
l_j i_j level marginal lower upper scale
1 seattle new-york 50 0.000 0 Inf 1
2 seattle chicago 300 0.000 0 Inf 1
3 seattle topeka 0 0.036 0 Inf 1
4 san-diego new-york 275 0.000 0 Inf 1
5 san-diego chicago 0 0.009 0 Inf 1
6 san-diego topeka 275 0.000 0 Inf 1
```

This syntax assumes that the user will always want to read in both the metadata as well as the actual data records, but it is possible to skip the reading of the records by passing the argument records=FALSE.

```R
m = Container$new()
> m$read("transport.gdx", "x", records = FALSE)
> names(m$data)
[1] "x"
> m$data$x$summary
$name
[1] "x"
$type
[1] "positive"
$domain_objects
$domain_objects[[1]]
[1] "i"
$domain_objects[[2]]
[1] "j"
$domainNames
[1] "x" "j"
$dimension
[1] 2
$description
[1] "shipment quantities in cases"
$numberRecords
[1] 0
$domainType
[1] "relaxed"
> m$data$x$records
NULL
```
Attention

The `read` method attempts to link the domain objects together (in order to have a "regular" domainType) but if domain sets are not part of the read operation there is no choice but to default to a "relaxed" domain type. This can be seen in the last example where we only read in the variable `x` and not the domain sets (`i` and `j`) that the variable is defined over. All the data will be available to the user, but domain checking is no longer possible. The symbol `x` will remain with "relaxed" domain type even if the user were to read in sets `i` and `j` in a second `read` call.

7.7.1.3.10.2 Writing to GDX

A user can write data to a GDX file by simply passing a file path (as a string). The `write` method will then create the GDX and write all data in the `Container`.

Note

It is not possible to write the `Container` when any of its symbols are invalid. If any symbols are invalid an error will be raised and the user will need to inspect the problematic symbols (perhaps using a combination of the `listSymbols(isValid=FALSE)` and `isValid(verbose=TRUE)` methods).

Example

```r
m$write("path/to/file.gdx")
```

Example (write a compressed GDX file)

```r
m$write("path/to/file.gdx", compress = TRUE)
```

Advanced users might want to specify an order to their UEL list (i.e., the universe set); recall that the UEL ordering follows that dictated by the data. As a convenience, it is possible to prepend the UEL list with a user specified order using the `uelPriority` argument.

Example (change the order of the UEL)

```r
a = Container$new()
i = Set$new(a, "i", records=c("a", "b", "c"))
m$write("foo.gdx", uelPriority=c("a", "c"))
```

The original UEL order for this GDX file would have been `c("a", "b", "c")`, but this example reorders the UEL with `uelPriority` – the positions of `b` and `c` have been swapped. This can be verified with the `gdxdump` utility (using the `uelTable` argument):

```sh
gdxdump foo.gdx ueltable=foo
```

Set foo /
  'a' ,
  'c' ,
  'b' /;
$onEmpty

Set i(*) /
  'a' ,
  'c' ,
  'b' /;
$offEmpty
7.8 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C# API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known transportation problem. These examples introduce several API features step by step.

- Getting Started A quick introduction about how to create and configure C# project
- Important Classes of the API Overview of some fundamental classes of the GAMS C# API
- How to use API An extensive set of examples how to use API components

7.8.1 Getting Started

This section takes you through the basic steps of creating and configuring a C# project in Visual Studio 2010 for use with the GAMS C# API. At the end of this section there is also a paragraph about Mono as an alternative to Microsoft Visual Studio.

7.8.1.1 Open a new Project

After opening Microsoft Visual Studio you can open the New Project wizard by choosing File > New > Project from the ribbon menu and type a name for your application, e.g. GAMSApplication.

7.8.1.2 Add a Reference to a Project

Once you opened the project add a reference by a right click on References inside the solution explorer. Then choose Add Reference.. and select the Browse tab. Next navigate to the location where you installed GAMS.net4.dll, typically the GAMS system directory, e.g. C:\GAMS\win64\40.1. Then choose GAMS.net4.dll and click OK.
7.8.1.3 Solving your first Model

This section illustrates how a model from the GAMS model library can be imported and solved.

Open a new project and reference the GAMS.net4.dll in your project as explained above. Open Program.cs which should be part of your project and replace the content by:

```csharp
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using GAMS;
namespace TransportSeq
{
    class Transport0
    {
        static void Main(string[] args)
        {
            GAMSWorkspace ws = new GAMSWorkspace();
            ws.GamsLib("trnsport");
            // create a GAMSJob from file and run it with default settings
            GAMSJob t0 = ws.AddJobFromFile("trnsport.gms");
            t0.Run();
            Console.WriteLine("Ran with Default:");
            foreach (GAMSVariableRecord rec in t0.OutDB.GetVariable("x"))
            {
            }
            Console.WriteLine("z= " + t0.OutDB.GetVariable("z").LastRecord().Level);
        }
    }
}
```

Then click the highlighted button from figure (a) down below or press F5 to start debugging. You can also start the program without debugging by pressing Control + F5 or choosing Debug > Start Without Debugging from the ribbon menu as illustrated in figure (b).
By following the latter instruction the command line shell remains open and show the produced output.

### 7.8.1.4 Mono

The GAMS .NET API works also on non-Windows platforms where no Visual Studio is available. It can be used for example with the open source .Net framework Mono ([http://www.mono-project.com](http://www.mono-project.com)). Mono offers a IDE that could be used similarly to Visual Studio. If one prefers to compile and execute from a shell, the following code shows how this can be done using the example Transport1 which comes with the GAMS system (this assumes that xbuild and mono are in the PATH):

```bash
cd <GAMS Dir>/apifiles/CSharp/Transport1
xbuild /t:rebuild /p:Configuration=Release Transport1.csproj
mono bin/Release/Transport1.exe ../..../..
```

### 7.8.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples.

- **GAMS Namespace**
- **GAMSWorkspace Class**
- **GAMSJob Class**
- **GAMSDatabase Class**
- **GAMSOptions Class**
- **GAMSModelInstance Class**

### 7.8.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C# API. `<GAMS system directory>/apifiles/CSharp` contains a file `TransportSeq.sln` that can be opened in Microsoft Visual Studio. The contained projects deal with the well-known transportation problem. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)
- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to specify the solver (Transport1)
- How to run a job with a solver option file and capture its log output (Transport1)
7.8.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing Transport1.exe with C:/GAMS/win64/40.1 we use the 64-bit version of GAMS 40.1 to run Transport1 even if our default GAMS system might be a different one. This is managed by the following code:

```csharp
GAMSWorkspace ws;
if (Environment.GetCommandLineArgs().Length > 1)
    ws = new GAMSWorkspace(systemDirectory: Environment.GetCommandLineArgs()[1]);
else
    ws = new GAMSWorkspace();
...
```

Remember that the bitness of the GAMS system has to match the bitness of your .NET program.

7.8.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented .NET API offers much more than exchanging data between .NET and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in .NET data structures. We assume that the data to be exported is available in .NET data structures.

```csharp
List<string> plants = new List<string>()
{  
    "Seattle", "San-Diego"
};
List<string> markets = new List<string>()
{
    "New-York", "Chicago", "Topeka"
};
Dictionary<string, double> capacity = new Dictionary<string, double>()
{
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
```
Different GAMS symbols are represented using different .NET data structures. The data for the GAMS sets is represented using lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with .NET data structures.

A new GAMSDatabase instance can be created using `GAMSWorkspace.AddDatabase`. ...

We start adding GAMS sets using the method `GAMSDatabase.AddSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A foreach-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.AddRecord`. ...

GAMSParameter instances can be added by using the method `GAMSDatabase.AddParameter`. It has the same signature as `GAMSDatabase.AddSet`. Anyhow, in this example we use an overload of the method which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

As soon as all data is prepared in the GAMSDatabase, the method `GAMSDatabase.Export` can be used to create a GDX file.

7.8.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.AddDatabaseFromGDX`. The method takes a path to a GDX file and creates a GAMSDatabase instance.

Reading the data from the GAMSSet `i` into a list can be done as follows:

7.8 Tutorial
A new list `plants2` is created. `i` is retrieved by calling `GAMSDatabase.GetSet` on `db2`. The returned `GAMSSet` object can be iterated using a `foreach`-loop to access the records of the set. Each record is of type `GAMSSetRecord` and can be asked for its keys.

You can do the same for `GAMSParameterRecords`. Instead of creating a list, we want to have the data in the form of a dictionary. `GAMSParameterRecords` can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `Dictionary<string, double>`.

```csharp
Dictionary<string, double> capacity2 = new Dictionary<string, double>();
foreach (GAMSParameterRecord item in db2.GetParameter("a"))
    capacity2.Add(item.Key(0), item.Value);
```

For multi-dimensional symbols, we choose the dictionary keys to be tuples instead of string.

```csharp
Dictionary<Tuple<string, string>, double> distance2 = new Dictionary<Tuple<string, string>, double>();
foreach (GAMSParameterRecord item in db2.GetParameter("d"))
    distance2.Add(new Tuple<string, string>(item.Key(0), item.Key(1)), item.Value);
```

### 7.8.3.4 How to run a GAMSJob from file (Transport1)

Here we load the model `transport` from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the `GAMSWorkspace.AddJobFromFile` Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the `GAMSJob` `t1` is defined from that file and run by the `GAMSJob.Run` method.

The following lines create the solution output and illustrate the usage of the `GAMSJob.OutDB` property to get access to the `GAMSDatabase` created by the `Run` method. To retrieve the content of variable `x` we use the `GAMSVariableRecord` class and the `GAMSDatabase.GetVariable` method.

```csharp
ws.GamsLib("transport");
// create a GAMSJob from file and run it with default settings
GAMSJob t1 = ws.AddJobFromFile("transport.gms");
t1.Run();
Console.WriteLine("Ran with Default:
foreach (GAMSVariableRecord rec in t1.OutDB.GetVariable("x"))
```

### 7.8.3.5 How to specify the solver (Transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.AddOptions` method. The `GAMSOptions.AllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle.

```csharp
using (GAMSOptions opt = ws.AddOptions())
{
    opt.AllModelTypes = "xpress";
    t1.Run(opt);
}
```
7.8.3.6 How to run a job with a solver option file and capture its log output (Transport1)

At first we use the StreamWriter to create the file xpress.opt which will be used as solver option file and is stored in the current working directory. Then we write algorithm=barrier to this file and specify the barrier algorithm as algorithm before we close the solver option file. We choose xpress as solver just like in the preceding example and set the GAMSOptions.OptFile Property to 1 to tell the solver to look for a solver option file. In addition, we specify the argument output in order to write the log of the GAMSJob into the file transport1.xpress.log.

```csharp
... using (StreamWriter optFile = new StreamWriter(Path.Combine(ws.WorkingDirectory, "xpress.opt")))
using (GAMSOptions opt = ws.AddOptions())
{
    optFile.WriteLine("algorithm=barrier");
    optFile.Close();
    opt.AllModelTypes = "xpress";
    opt.OptFile = 1;
    t1.Run(opt, output: logFile);
}
... Instead of writing the log output to a file, it can be written into any TextWriter. In order to write the log directly to Console.Out, we can use the following code:
... t1.Run(opt, output: Console.Out);
...

7.8.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings GetDataText and GetModelText are using GAMS syntax.

At first we write an include file tdata.gms that contains the data but not the model text:

```csharp
... using (StreamWriter writer = new StreamWriter(ws.WorkingDirectory + Path.DirectorySeparatorChar + "tdata.gms"))
{
    writer.Write(GetDataText());
}
...
```

Afterwards we create a GAMSJob using the GAMSWorkspace.AddJobFromString method. The GAMSOptions.Defines field is used like the 'double dash' GAMS parameters, i.e. it corresponds to --incname=tdname on the command line.

```csharp
... using (GAMSOptions opt = ws.AddOptions())
{
    GAMSJob t2 = ws.AddJobFromString(GetModelText());
    opt.Defines.Add("incname", "tdata");
    t2.Run(opt);
}
...
```

Note that the string GetModelText contains the following lines to read in the data.

```csharp
... $if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.8.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string GetDataText as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding GAMSJob a GAMSDatabase is created that is available via the GAMSJob.OutDb property. We can use the GAMSDatabase.Export method to write the content of this database to a gdx file tdata.gdx.

```csharp
GAMSJob t3 = ws.AddJobFromString(GetDataText());
t3.Run();
...
7.8.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in Transport2 - How to use include files (Transport2).

```csharp
... t3 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
    opt.Defines.Add("gdxincname", "tdata");
    opt.AllModelTypes = "xpress";
    t3.Run(opt);
    ...
}
...
```

Note that there are some changes in GetModelText due to the usage of a GDX file instead of an include file.

```csharp
... $if not set gdxincname $abort 'no include file name for data file provided'
.gdxin %gdxincname%
load i j a b d f
.gdxin ...
```

7.8.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding `OutDB` can be read in directly just like a gdx file. Note that the database needs to be passed to the `GAMSJob.Run` method as additional argument.

```csharp
... using (GAMSOptions opt = ws.AddOptions())
{
    GAMSJob t3a = ws.AddJobFromString(GetDataText());
    GAMSJob t3b = ws.AddJobFromString(GetModelText());
    t3a.Run();
    opt.Defines.Add("gdxincname", t3a.OutDB.Name);
    opt.AllModelTypes = "xpress";
    t3b.Run(opt, t3a.OutDB);
    ...
}
...
```

7.8.3.11 How to define data using C# data structures (Transport4)

We use the `List<T>` class, the `Dictionary<TKey, TValue>` class and the `Tuple<T1, T2>` class to define C# data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```csharp
List<string> plants = new List<string>()
{ "Seattle", "San-Diego" };
List<string> markets = new List<string>()
{ "New-York", "Chicago", "Topeka" };
Dictionary<string, double> capacity = new Dictionary<string, double>()
{ { "Seattle", 350.0 }, { "San-Diego", 600.0 } };
Dictionary<string, double> demand = new Dictionary<string, double>()
{ { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 } };
Dictionary<Tuple<string,string>, double> distance = new Dictionary<Tuple<string,string>, double>()
{ new Tuple<string,string>("Seattle", "New-York"), 2.5 },
 new Tuple<string,string>("Seattle", "Chicago"), 1.7 ],
 new Tuple<string,string>("Seattle", "Topeka"), 1.8 ],
 new Tuple<string,string>("San-Diego", "New-York"), 2.5 ],
 new Tuple<string,string>("San-Diego", "Chicago"), 1.8 ],
 new Tuple<string,string>("San-Diego", "Topeka"), 1.4 ]
};
...
7.8 Tutorial

7.8.3.12 How to prepare a GAMSDatabase from C# data structures (Transport4)

At first we create an empty GAMSDatabase db using the `GAMSWorkspace.AddDatabase` method. Afterwards we prepare the database. To add a set to the database we use the GAMSSet class and the `GAMSDatabase.AddSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C# data structure and add them by using the `GAMSSet.AddRecord` method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The GAMSJob can be run like explained in the preceding example about implicit database communication.

```csharp
GAMSDatabase db = ws.AddDatabase();
GAMSSet i = db.AddSet("i", 1, "canning plants");
foreach (string p in plants)
  i.AddRecord(p);
...
GAMSParameter a = db.AddParameter("a", 1, "capacity of plant i in cases");
foreach (string p in plants)
  a.AddRecord(p).Value = capacity[p];
...
GAMSParameter d = db.AddParameter("d", 2, "distance in thousands of miles");
foreach (Tuple<string, string> t in distance.Keys)
  d.AddRecord(t.Item1, t.Item2).Value = distance[t];
GAMSParameter f = db.AddParameter("f", 0, "freight in dollars per case per thousand miles");
  f.AddRecord().Value = 90;
// run a job using data from the created GAMSDatabase
GAMSJob t4 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
  opt.Defines.Add("gdxincname", db.Name);
  opt.AllModelTypes = "xpress";
  t4.Run(opt, db);
}
...
```

7.8.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS Checkpoint, the second one uses the `GAMSWorkspace.AddJobFromString` method to create a GAMSJob containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.Run` method. Furthermore, it passes an additional checkpoint argument to the Run method. That means the GAMSCheckpoint cp captures the state of the GAMSJob.

```csharp
GAMSCheckpoint cp = ws.AddCheckpoint();
...
ws.AddJobFromString(GetModelText()).Run(cp);
...
```

This creates the same checkpoint as for example the following code snippet:

```csharp
GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t5a = ws.AddJobFromString(GetModelText());
t5a.Run(cp);
```

7.8.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `GetModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```csharp
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j); 
...
Scalar ms 'model status', ss 'solve status'; 
...
We create a list with eight different values for this demand multiplier.

double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };

For each entry of that list we create a GAMSJob `t5` using the `GAMSWorkspace.AddJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a GAMSJob combined from the checkpoint plus the content provided by the string.

We run the GAMSJob and echo some interesting data from the OutDB using the `GAMSDatabase.GetParameter` and `GAMSDatabase.GetVariable` methods, the `GAMSParameter.FindRecord` and `GAMSVariable.FindRecord` methods plus the `GAMSParameterRecord.Value` property and the `GAMSVariableRecord.Level` property.

```csharp
foreach (double b in bmultlist) {
    GAMSJob t5 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
    ss=transport.solvestat;", cp);
    t5.Run();
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine(" Modelstatus: " + t5.OutDB.GetParameter("ms").FindRecord().Value);
    Console.WriteLine(" Solvestatus: " + t5.OutDB.GetParameter("ss").FindRecord().Value);
    Console.WriteLine(" Obj: " + t5.OutDB.GetVariable("z").FindRecord().Level);
}
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

### 7.8.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint

(Transport6)

This example illustrates how to run the jobs known from `Transport5` in parallel. We initialize the GAMSCheckpoint `cp` and introduce a demand multiplier as we did before:

```csharp
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
```

Furthermore, we introduce a new object `ioMutex` that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the list of demand multipliers we call a delegate of the function `RunScenario`. Note that these calls are parallel!

```csharp
// run multiple parallel jobs using the created GAMSCheckpoint
Object ioMutex = new Object();
System.Threading.Tasks.Parallel.ForEach(bmultlist, delegate(double b) {
    RunScenario(ws, cp, ioMutex, b);
});
```

In function `RunScenario` a GAMSJob is created and run just like in the preceding example of `Transport5`. The output section is also the same except for the fact that it is marked as critical section by the `lock` keyword. That means the delegates of `RunScenario` are running in parallel but the output block of different delegates cannot be executed in parallel since it is 'locked' by the same object `ioMutex` for all delegates.

```csharp
static void RunScenario(GAMSWorkspace ws, GAMSCheckpoint cp, object ioMutex, double b)
{
    GAMSJob t6 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
    ss=transport.solvestat;", cp);
    t6.Run();
    // we need to make the output a critical section to avoid messed up report information
    lock (ioMutex)
    {
        Console.WriteLine("Scenario bmult=" + b + ":");
        Console.WriteLine(" Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
        Console.WriteLine(" Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
        Console.WriteLine(" Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
    }
}
While the output in Transport5 is strictly ordered subject to the order of the elements of \texttt{bmultlist}, in Transport6 the output blocks might change their order but the blocks describing one scenario are still appearing together due to the \texttt{lock} keyword.

If you want a further impression of the impact of the \texttt{lock} keyword, just rerun Transport6 but comment out the \texttt{lock} as follows and compare the output.

```csharp
//lock (ioMutex)  
//
//lock (ioMutex)  
//
//lock (ioMutex)  
//
Console.WriteLine("Scenario bmult=" + b + ":");
Console.WriteLine(" Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
Console.WriteLine(" Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
Console.WriteLine(" Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
//
```

### 7.8.3.16 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In Transport7 the usage of \texttt{GAMS::GAMSModelInstance} is demonstrated.

At first checkpoint \texttt{cp} is created as in the preceding examples. Then we create the GAMSModelInstance \texttt{mi} using the \texttt{GAMSCheckpoint.AddModelInstance} method. Note that the \texttt{GAMSJob} again contains no solve statement and the demand multiplier is already included with default value 1.

```csharp
GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t7 = ws.AddJobFromString(GetModelText());
t7.Run(cp);
GAMSModelInstance mi = cp.AddModelInstance();
```

### 7.8.3.17 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A GAMSModelInstance uses a SyncDB to maintain the data. We define \texttt{bmult} as \texttt{GAMSParameter} using the \texttt{GAMSDatabase.AddParameter} method and specify \texttt{gurobi} as solver. Afterwards the GAMSModelInstance is instantiated with arguments \texttt{opt} and \texttt{GAMSModifier bmult}. The \texttt{GAMSModifier} means that \texttt{bmult} is modifiable while all other parameters, variables and equations of ModelInstance \texttt{mi} stay unchanged. We use the \texttt{GAMSParameter.AddRecord} method to assign a value to \texttt{bmult} that can be varied afterwards using the \texttt{GAMSParameter.FirstRecord} method to reproduce our well-known example with different demand multipliers.

```csharp
GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.AddOptions();
opt.AllModelTypes = "gurobi";
mi.Instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
bmult.AddRecord().Value = 1.0;
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
foreach (double b in bmultlist)
{
    bmult.FirstRecord().Value = b;
    mi.Solve();
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine(" Modelstatus: " + mi.ModelStatus);
    Console.WriteLine(" Solvestatus: " + mi.SolveStatus);
    Console.WriteLine(" Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
}
```
7.8.3.18 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a GAMSModelInstance just like in the next to last example. We define \( x \) as GAMSVariable and its upper bound as GAMSParameter \( xup \). At the following instantiate method GAMSModifier has three arguments. The first one says that \( x \) is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GAMSParameter that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```csharp
mi = cp.AddModelInstance();
GAMSVariable x = mi.SyncDB.AddVariable("x", 2, VarType.Positive, "");
GAMSParameter xup = mi.SyncDB.AddParameter("xup", 2, "upper bound on x");
mi.Instantiate("transport use lp min z", modifiers: new GAMSModifier(x, UpdateAction.Upper, xup));
foreach (GAMSSetRecord i in t7.OutDB.GetSet("i"))
    foreach (GAMSSetRecord j in t7.OutDB.GetSet("j"))
    {
        xup.Clear();
        xup.AddRecord(i.Keys[0], j.Keys[0]).Value = 0;
        mi.Solve();
        Console.WriteLine("Scenario link blocked: " + i.Keys[0] + " - " + j.Keys[0]);
        Console.WriteLine(" Modelstatus: " + mi.ModelStatus);
        Console.WriteLine(" Solvestatus: " + mi.SolveStatus);
        Console.WriteLine(" Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
    }
```

7.8.3.19 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint \( cp \) from a GAMSJob. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out-principle. The objects queueMutex and ioMutex are used later to avoid messed up output. Then we call two delegates of the function ScenSolve in parallel that get the same queue as argument.

```csharp
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);
Queue<double> bmultQueue = new Queue<double>(new double[]{0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3});
Object queueMutex = new Object();
Object ioMutex = new Object();
Parallel.For(0, 2, delegate(int i)
    {
        ScenSolve(ws, cp, bmultQueue, queueMutex, ioMutex);
    });
```

In function ScenSolve we create and instantiate a GAMSModelInstance as in the preceding examples and make bmult modifiable. The two delegates of the function extract the elements of the queue using bmultQueue.Dequeue. Note that we chose cplex as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```csharp
private static void ScenSolve(GAMSWorkspace ws, GAMSCheckpoint cp, Queue<double> bmultQueue, Object queueMutex, Object ioMutex, int i)
{
    GAMSModelInstance mi = cp.AddModelInstance();
    GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
    GAMSOptions opt = ws.AddOptions();
    opt.AllModelTypes = "cplex";
    mi.Instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
    bmult.AddRecord().Value = 1.0;
    while (true)
    {
        double b;
        // dynamically get a bmult value from the queue instead of passing it to the different threads at creation time
        lock (queueMutex)
        {
            if (0 == bmultQueue.Count)
            {
                return;
            }
            b = bmultQueue.Dequeue();
        }
        bmult.FirstRecord().Value = b;
        mi.Solve();
        // we need to make the output a critical section to avoid messed up report informations
    }
}
```
7.8 Tutorial

lock (ioMutex)
{
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine(" Modelstatus: "+mi.ModelStatus);
    Console.WriteLine(" Solvestatus: "+mi.SolveStatus);
    Console.WriteLine(" Obj: "+mi.SyncDB.GetVariable("z").FindRecord().Level);
}
}

7.8.3.20 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a GAMSDatabase. We call a function ReadFromAccess that finally returns a GAMSDatabase as shown below.

...GAMSDatabase db = ReadFromAccess(ws);
...

The data we are going to read can be found in <GAMS system directory>\apidfiles\Data\transport.accdb. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty GAMSDatabase. Afterwards we connect to the MS Access database by first specifying the provider and then defining the aforementioned file as data source. The OleDbConnection represents an open connection to a data source. The following lines initialize the connection and use a try and a catch block for potential exceptions, i.e. errors that occur during application execution. To finally read in GAMS sets and parameters we call the functions ReadSet and ReadParameter. We will use several classes of the System.Data.OleDb namespace that are documented here. Furthermore, we recommend the OLE DB Tutorial.

```csharp
static GAMSDatabase ReadFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.AddDatabase();
    // connect to database
    string strAccessConn = ";Provider=Microsoft.ACE.OLEDB.12.0;Data Source=..\..\..\..\..\..\..\..\..\Data\transport.accdb;"
    OleDbConnection connection = null;
    try
    {
        connection = new OleDbConnection(strAccessConn);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: Failed to create a database connection. 
    Environment.Exit(1);
    }
    // read GAMS sets
    ReadSet(connection, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
    ReadSet(connection, db, "SELECT Market FROM Market", "j", 1, "markets");
    // read GAMS parameters
    ReadParameter(connection, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
    ReadParameter(connection, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
    ReadParameter(connection, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles");
    return db;
}

The function ReadSet incorporates a try and a catch block. In the try block we prepare the reading from the MS Access file. Then we add a set symbol to the GAMSDatabase that is filled with the data from the MS Access file afterwards. The function ReadParameter works quite similar.

```csharp
static void ReadSet(OleDbConnection connect, GAMSDatabase db, string strAccessSelect, string setName, int setDim, string setExp = "")
{
    try
    {
        OleDbCommand cmd = new OleDbCommand(strAccessSelect, connect);
        connect.Open();
        OleDbDataReader reader = cmd.ExecuteReader();
        if (reader.FieldCount != setDim)
        {
            Console.WriteLine("Number of fields in select statement does not match setDim");
            Environment.Exit(1);
        }
        GAMSSet i = db.AddSet(setName, setDim, setExp);
        for (int idx = 0; idx < setDim; idx++)
        {
            keys[idx] = reader.GetString(idx);  // make sure keys[idx] is not null
        }
    }
    ```
catch (Exception ex)
{
    Console.WriteLine("Error: Failed to retrieve the required data from the Database.\n[0]", ex.Message);
    Environment.Exit(1);
}
finally
{
    connect.Close();
}
}

Once we read in all the data we can create a GAMSJob from the GAMSDatabase and run it as usual.

7.8.3.21 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `<GAMS system directory>\apifiles\Data\transport.xlsx`. The model is given as string without data like in many examples before. At first we have to add

```csharp
using Excel = Microsoft.Office.Interop.Excel;
```

to the preamble to be able to use the Microsoft.Office.Interop.Excel namespace. Then we define excelApp as Excel application using the implicit variable type var, use the aforementioned file as workbook wb and define the Excel range that can represent a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells, or a 3-D range.

```csharp
var excelApp = new Excel.Application();
Excel.Range range;
```

The following lines address the different worksheets and read in the contained data. Afterwards we make an errorcheck to ensure that the number of plants and markets is the same in all worksheets.

```csharp
Excel.Worksheet capacity = (Excel.Worksheet)wb.Worksheets.get_Item("capacity");
range = capacity.UsedRange;
Array capacityData = (Array)range.Cells.Value;
int iCount = capacity.UsedRange.Columns.Count;
Excel.Worksheet demand = (Excel.Worksheet)wb.Worksheets.get_Item("demand");
range = demand.UsedRange;
Array demandData = (Array)range.Cells.Value;
int jCount = range.Columns.Count;
Excel.Worksheet distance = (Excel.Worksheet)wb.Worksheets.get_Item("distance");
range = distance.UsedRange;
Array distanceData = (Array)range.Cells.Value;
// number of markets/plants have to be the same in all spreadsheets
Debug.Assert((range.Columns.Count - 1) == jCount && (range.Rows.Count - 1) == iCount,
    "Size of the spreadsheets doesn't match");
wb.Close();
```

If you have problems to see through the steps above, adding the following output section right after `wb.Close()` might be helpful to understand where which data is stored.

```csharp
for (int ii = 1; ii <= iCount; ii++)
{
    Console.WriteLine("capacityData(1, " + ii + ") = " + capacityData.GetValue(1, ii));
    Console.WriteLine("capacityData(2, " + ii + ") = " + capacityData.GetValue(2, ii));
}
for (int jj = 1; jj <= jCount; jj++)
{
    Console.WriteLine("demandData(1, " + jj + ") = " + demandData.GetValue(1, jj));
    Console.WriteLine("demandData(2, " + jj + ") = " + demandData.GetValue(2, jj));
}
for (int ii = 1; ii <= iCount; ii++)
{
    for (int jj = 1; jj <= jCount; jj++)
    {
        Console.WriteLine("distanceData(" + (ii+1) + "," + (jj+1) + ") = " + distanceData.GetValue(ii+1, jj+1));
    }
}
```

Now we can create the GAMSWorkspace as usual and afterwards create a GAMSDatabase and fill it with the workbook data as follows:
...  
GAMSDatabase db = ws.AddDatabase();  
GAMSSet Set1 = db.AddSet("i", 1, "Plants");  
GAMSSet Set2 = db.AddSet("j", 1, "Markets");  
GAMSParameter capacityParam = db.AddParameter("a", 1, "Capacity");  
GAMSParameter demandParam = db.AddParameter("b", 1, "Demand");  
GAMSParameter distanceParam = db.AddParameter("d", 2, "Distance");  
for (int ic = 1; ic <= iCount; ic++)  
{  
    Set1.AddRecord((string)capacityData.GetValue(1, ic));  
    capacityParam.AddRecord((string)capacityData.GetValue(1, ic)).Value = (double)capacityData.GetValue(2, ic);  
}  
for (int jc = 1; jc <= jCount; jc++)  
{  
    j.AddRecord((string)demandData.GetValue(1, jc));  
    demandParam.AddRecord((string)demandData.GetValue(1, jc)).Value = (double)demandData.GetValue(2, jc);  
}  
for (int ic = 1; ic <= iCount; ic++)  
{  
    distanceParam.AddRecord((string)distanceData.GetValue(ic + 1, 1), (string)distanceData.GetValue(1, jc + 1)).Value =  
        (double)distanceData.GetValue(ic + 1, jc + 1);  
}  
...  
Note that we can name sets and parameters just like in the database but we don't have to. Now we finally  
set up the database and can run our GAMSJob as usual.  
...  
using (GAMSOptions opt = ws.AddOptions())  
{  
    GAMSJob t10 = ws.AddJobFromString(GetModelText());  
    opt.Defines.Add("gdxincname", db.Name);  
    opt.AllModelTypes = "xpress";  
    t10.Run(opt, db);  
    foreach (GAMSVariableRecord rec in t10.OutDB.GetVariable("x"))  
    {  
    }  
}  
...  
7.8.3.22 How to create and use a save/restart file (Transport11)  

In Transport11 we demonstrate how to create and use a save/restart file. Usually such a file should be  
supplied by an application provider but in this example we create one for demonstration purpose. Note  
that the restart is launched from a GAMSCheckpoint.  

In the main function we start with the creation of a folder called tmp that internally is denoted as wDir.  
Then we call the function CreateSaveRestart.  
...  
string wDir = Path.Combine(\".\", "tmp\");  
CreateSaveRestart(Path.Combine(wDir, "tbase\"));  
...  
In function CreateSaveRestart we choose the path of the file given as argument as working directory,  
so it should be wDir. Then we create a GAMSJob from a string. Note that the string given via  
GetBaseModelText() contains the basic definitions of sets without giving them a content (that is what  
$onempty is used for). Afterwards we specify a GAMSOption to only compile the job but do not execute  
it. Then we create a checkpoint cp that is initialized by the following run of the GAMSJob and stored in  
the file given as argument to the function. This becomes possible because the AddCheckpoint method  
accepts identifiers as well as file names as argument.  
static void CreateSaveRestart(string cpFileName)  
{  
    GAMSWorkspace ws;  
    if (Environment.GetCommandLineArgs().Length > 1)  
        ws = new GAMSWorkspace(workingDirectory: Path.GetDirectoryName(cpFileName), systemDirectory:  
            Environment.GetCommandLineArgs()[1]);  
    else  
        ws = new GAMSWorkspace(workingDirectory: Path.GetDirectoryName(cpFileName));  
    GAMSJob j1 = ws.AddJobFromString(GetBaseModelText());  
    GAMSOptions opt = ws.AddOptions();  
    opt.Action = GAMSOptions.EAction.CompileOnly;  
    GAMSCheckpoint cp = ws.AddCheckpoint(Path.GetFileName(cpFileName));  
    j1.Run(opt, cp);  
    opt.Dispose();  
}
So what you should keep in mind before we return to further explanations of the main function is, that now the file thase in folder tmp which is denoted as wDir contains a checkpoint. Now in the main function we define some data using C# data structures as we already did in Transport4 before we create the GAMSWorkspace with working directory wDir.

```csharp
GAMSWorkspace ws;
if (Environment.GetCommandLineArgs().Length > 1)
    ws = new GAMSWorkspace(workingDirectory: wDir, systemDirectory: Environment.GetCommandLineArgs()[1]);
else
    ws = new GAMSWorkspace(workingDirectory: wDir);
...
```

Afterwards we set up the GAMSDatabase like we already did in Transport4. Once this is done we run a GAMSJob using this data plus the checkpoint stored in file thase.

```csharp
GAMSCheckpoint cpBase = ws.AddCheckpoint("tbase");
using (GAMSOptions opt = ws.AddOptions())
    {
    GAMSJob t4 = ws.AddJobFromString(GetModelText(), cpBase);
    opt.Defines.Add("gdxincname", db.Name);
    opt.AllModelTypes = "xpress";
    t4.Run(opt, db);
    foreach (GAMSVariableRecord rec in t4.OutDB.GetVariable("x"))
    }
...```

Note that the string from which we create job t4 is different to the one used to prepare the checkpoint stored in tbase and is only responsible for reading in the data from the GAMSDatabase correctly. The entire model definition is delivered by the checkpoint cpBase which is equal to the one we saved in tbase.

## 7.9 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C++ API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known transportation problem. These examples introduce several API features step by step. The GAMS distribution comes with a pre compiled binary version of the GAMS C++ API on Windows, Mac OS X and Linux. Futhermore, the C++ API itself and the provided examples are published under MIT license and are hosted at the GAMS GitHub organization. Therefore it is possible to compile the library manually (e.g. if a certain compiler is required) The GAMS C++ API is implemented and compatible with C++17.

- **Getting Started** A quick introduction about how to create and configure a C++ project using Qt Creator
- **Important Classes of the API** Overview of some fundamental classes of the GAMS C++ API
- **How to use API** An extensive set of examples how to use API components

### 7.9.1 Getting Started

This section guides you through the basic steps of compiling and running a program from scratch using either Qt Creator or Microsoft Visual Studio. If you want to have a look at the API itself you can directly start with the How to use API section instead. Note that you need to use at least C++17 in order to be able to use the GAMS C++ API. While this tutorial uses Qt Creator and Microsoft Visual Studio, other IDE's can be used as well.

Note

When using the pre compiled binary distribution of the GAMS C++ API as done in this section, one needs to use a compiler that is compatible with the one that was used during compilation of the library. Depending on the platform, different versions of the pre build binaries can be found under `<GAMS system directory>/apifiles/C++/lib`. In GAMS versions before 25.0 the binaries can be found under `<GAMS system directory>/`. 
7.9 Tutorial

7.9.1.1 Getting Started using Qt Creator

After opening Qt Creator click on File > New File or Project. Choose Non-Qt Project > Plain C++ Application and click on Choose.... On the next page fill in the name and the location of the new project and click Next. Select the kits you want to use for the project and click again on Next. If you use the precompiled library on Windows, you need to use MSVC2017 or MSVC2019. Confirm the creation of the new project by clicking on Finish.

As soon as the project is created open the file GAMSApplication.pro by double clicking on it in the pane on the left hand side. Add the following two lines to the .pro file:

unix: LIBS += -ldl -L $$\$(HOME)/gams36.0/apifiles/C++/lib -lgamscpp
win32: LIBS += -LC:\GAMS\36\apifiles\C++\lib\vs2019 -lgamscpp
INCLUDEPATH += C:\GAMS\36\apifiles\C++\api

Note

Depending on your operating system and the location of your GAMS system you need to adjust the paths respectively. Furthermore, all pre build binaries are always located at <GAMS system directory>\apifiles\C++\lib, if GAMS 25.0 (or higher) is used. This location may contain additional sub directories, like <GAMS system directory>\apifiles\C++\lib\vs2019. In case a previous version of GAMS is used the binaries are located at <GAMS system directory>.

Save the changes, right click on the project and choose Run qmake.

The last step of creating a first small example is to write code that makes use of the GAMS C++ API. Open the file main.cpp and replace the content of the file with the following lines of code:

```cpp
#include "gams.h"
#include <iostream>
using namespace gams;
using namespace std;
int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("transport");
    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("transport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << "," << rec.key(1) << ")=" << rec.level() << " marginal=" << rec.marginal() << endl;
}
```

Make sure that MSVC2019 64bit is chosen as kit using the Release mode.

Click on the play button below to run (and compile) the small example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

7.9.1.2 Getting Started using Visual Studio 2019

After opening Visual Studio click on File > New > Project ....

Choose Console App and fill in the name and the location of the project and click on Ok.

As soon as the project is loaded, open the Configuration Manager:

Change the Active solution configuration to Release and the Active solution platform to x64.

The next step is to configure the project to find the GAMS C++ API. Right click on the project and choose Properties from the context menu. Go to Configuration Properties > C/C++ > General and add <GAMS system directory>\apifiles\C++\api to Additional Include Directories.

Select Configuration Properties > C/C++ > Precompiled Headers on the left. On the right open the dropdown of Precompiled Header and select Not Using Precompiled Headers.

Go to Configuration Properties > Linker > General and add the apifiles path to the matching version of your Visual Studio (e.g. <GAMS system directory>\apifiles\C++\lib\vs2017) to Additional Library Directories.
Note

Please mind that the pre build binaries for older GAMS versions (before GAMS 25.0) are located at <GAMS system directory>.

Choose Input and add gamscpp.lib to Additional Dependencies. Make sure that all previous entries including %(AdditionalDependencies) are present as well.

The last required configuration step is to add a post-build event that copies required files next to the generated executable. Click on Build Events > Post-Build Event and add the following code to Command Line. It is helpful to select <Edit...> from the dropdown box as there are two command lines. Note that you might need to adjust the used path depending on the location of your GAMS installation:

```
xcopy /Y "C:\gams\36\apifiles\C++\lib\vs2017\gamscpp.dll" "$\{OutDir\}"
```

Open the file GAMSApplication.cpp and replace its content by the following code:

```cpp
#include "gams.h"
#include <iostream>
using namespace gams;
using namespace std;
int main(int argc, char* argv[])
{
  GAMSWorkspace ws;
  ws.gamsLib("transport");
  // create a GAMSJob from file and run it with default settings
  GAMSJob t1 = ws.addJobFromFile("transport.gms");
  t1.run();
  for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
    cout << "x(" << rec.key(0) << "," << rec.key(1) << "): level=" << rec.level() << " marginal=" << rec.marginal() << endl;
}
```

Click on Debug > Start Without Debugging in order to compile and run the example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

### 7.9.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the gams namespace. Their usage is demonstrated by an extensive set of examples.

- **gams** Namespace
- **gams::GAMSWorkspace** Class
- **gams::GAMSJob** Class
- **gams::GAMSDatabase** Class
- **gams::GAMSOptions** Class
- **gams::GAMSMModelInstance** Class
- **gams::GAMSSymbol** Class
7.9 Tutorial

7.9.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C++ API. `<GAMS system directory>\apifiles\C++` contains the Qt project examples.pro and several Visual Studio solutions for different versions of Visual Studio (e.g. `examples-vs2017.sln`, `examples-vs2019.sln`). The code snippets that are explained in this tutorial are taken from these examples. Furthermore there is a `CMakeLists.txt` file that can be used for building the examples using CMake. The well-known transportation problem is discussed step by step and each example introduces new elements of the GAMS C++ API.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)
- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to specify the solver (Transport1)
- How to run a job with a solver option file (Transport1)
- How to use include files (Transport2)
- How to read data from string and export to GDX (Transport3)
- How to run a job using data from GDX (Transport3)
- How to run a job using implicit database communication (Transport3)
- How to define data using C++ data structures (Transport4)
- How to prepare a GAMSDatabase from C++ data structures (Transport4)
- How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)
- How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)
- How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)
- How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)
- How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)
- How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)
- How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)
- How to fill a GAMSDatabase by reading from MS Access (Transport9)
- How to fill a GAMSDatabase by reading from MS Excel (Transport10)
- How to create and use a save/restart file (Transport11)
7.9.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing Transport1.exe with C:/GAMS/40 we use the 64-bit version of GAMS 40.1 to run Transport1 even if our default GAMS system might be a different one. This is managed by the following code:

```cpp
... 
GAMSWorkspaceInfo wsInfo;
if (argc > 1) 
wsInfo.setSystemDirectory(argv[1]);
GAMSWorkspace ws(wsInfo);
... 
```

Remember that the bitness of the GAMS system has to match the bitness of your C++ program.

7.9.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented C++ API offers much more than exchanging data between C++ and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in C++ data structures.

```cpp
... 
vector<string> plants = { 
    "Seattle", "San-Diego"
};
vector<string> markets = { 
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = { 
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = { 
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = { 
    make_tuple("Seattle", "New-York"), 2.5,
    make_tuple("Seattle", "Chicago"), 1.7,
    make_tuple("Seattle", "Topeka"), 1.8,
    make_tuple("San-Diego", "New-York"), 2.5,
    make_tuple("San-Diego", "Chicago"), 1.8,
    make_tuple("San-Diego", "Topeka"), 1.4
};
... 
```

Different GAMS symbols are represented using different C++ data structures. The data for the GAMS sets is represented using vectors of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by maps (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with C++ data structures.

A new GAMSDatabase instance can be created using GAMSWorkspace.addDatabase.

```cpp
... 
// create new GAMSDatabase instance 
GAMSDatabase db = ws.addDatabase();
... 
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.addRecord`.

```cpp
// add 1-dimensional set 'i' with explanatory text 'canning plants' to the GAMSDatabase 
GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants) 
i.addRecord(p);
... 
```

GAMSPParameter instances can be added by using the method `GAMSDatabase.addParameter`. It has the same signature as `GAMSDatabase.addSet`. Anyhow, in this example we use an overload of the method.
which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

```plaintext
// add parameter 'a' with domain 'i'
GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (string p: plants)
  a.addRecord(p).setValue(capacity[p]);
```

As soon as all data is prepared in the GAMSDatabase, the method GAMSDatabase.doExport can be used to create a GDX file.

```plaintext
// export the GAMSDatabase to a GDX file with name 'data.gdx' located in the 'workingDirectory' of the GAMSWorkspace
db.doExport("data.gdx");
```

### 7.9.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.addDatabaseFromGDX`. The method takes a path to a GDX file and creates a GAMSDatabase instance.

```plaintext
// add a new GAMSDatabase and initialize it from the GDX file just created
GAMSDatabase db2 = ws.addDatabaseFromGDX("data.gdx");
```

Reading the data from the GAMSSet `i` into a vector can be done as follows:

```plaintext
// read data from symbols into C++ data structures
vector<string> iNew;
for (GAMSSetRecord rec : db2.getSet("i"))
  iNew.push_back(rec.key(0));
```

A new vector `iNew` is created. `i` is retrieved by calling `GAMSDatabase.getSet` on `db2`. The returned GAMSSet object can be iterated using a for-loop to access the records of the set. Each record is of type GAMSSetRecord and can be asked for its keys.

You can do the same for GAMSParameters. Instead of creating a vector, we want to have the data in the form of a map. GAMSParameterRecords can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `map<string, double>`.

```plaintext
map<string, double> aNew;
for (GAMSParameterRecord rec : db2.getParameter("a"))
  aNew[rec.key(0)] = rec.value();
```

For multi dimensional symbols, we choose the map keys to be tuples instead of string. We access the individual keys by index and generate a tuple using `make_tuple`.

```plaintext
map<tuple<string, string>, double> dNew;
for (GAMSParameterRecord rec : db2.getParameter("d"))
  dNew[make_tuple(rec.key(0), rec.key(1))] = rec.value();
```

Scalars can be read into a variable of type double by accessing the value of the first and only record.

```plaintext
double fNew = db2.getParameter("f").firstRecord().value();
```
7.9.3.4 How to run a GAMSJob from file (Transport1)

At first we create a GAMSWorkspace. Afterwards we load the model trnspor from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the GAMSWorkspace.AddJobFromFile Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GAMSJob t1 is defined from that file and run by the GAMSJob.run method. The following lines create the solution output and illustrate the usage of the GAMSJob.outDB method to get access to the GAMSDatabase created by the run method. To retrieve the content of variable x we use the GAMSVariableRecord class and the GAMSDatabase.getVariable method.

```cpp
... GAMSWorkspace ws; ws.gamsLib("trnsport"); // create a GAMSJob from file and run it with default settings GAMSJob t1 = ws.addJobFromFile("trnsport.gms"); // Default run t1.run(); cout << "Ran with Defaults:" << endl; for (GAMSVariableRecord rec : t1.outDB().getVariable("x")) cout << "x(" << rec.key(0) << "," << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" << rec.marginal() << endl; ...
```

7.9.3.5 How to specify the solver (Transport1)

The solver can be specified via the GAMSOptions class and the GAMSWorkspace.addOptions method. The GAMSOptions.setAllModelTypes method sets xpress as default solver for all model types which the solver can handle.

```cpp
... // Run the job again with another solver GAMSOptions opt = ws.addOptions(); opt.setAllModelTypes("xpress"); t1.run(opt); ...
```

7.9.3.6 How to run a job with a solver option file (Transport1)

At first create the file xpress.opt with content algorithm=barrier which will be used as solver option file and is stored in the current working directory. We choose xpress as solver just like in the preceding example and call GAMSOptions.setOptFile in order to tell the solver to look for a solver option file.

```cpp
... // Run the job with a solver option file ofstream xpressopt(ws.workingDirectory() + cPathSep + "xpress.opt"); xpressopt << "algorithm=barrier" << endl; xpressopt.close(); opt.setAllModelTypes("xpress"); opt.setOptFile(1); t1.run(opt); ...
```

7.9.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings provided by the methods getDataText and getModelText are using GAMS syntax.

At first we write an include file tdata.gms that contains the data but not the model text:

```cpp
... ofstream tdata(ws.workingDirectory() + cPathSep + "tdata.gms"); tdata << getDataText(); tdata.close(); ...
```
Afterwards we create a GAMSJob using the GAMSWorkspace.addJobFromString method. The GAMSOptions.setDefine method is used like the 'double dash' GAMS parameters, i.e. it corresponds to --incname=tdata on the command line.

```java
GAMSOptions opt = ws.addOptions();
GAMSJob t2 = ws.addJobFromString(getModelText());
opt.setDefine("incname", "tdata");
t2.run(opt);
...
```

Note that the string provided by getModelText contains the following lines to read in the data.

```java
if not set incname abort 'no include file name for data file provided'
@end
```

### 7.9.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string provided by getDataText as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding GAMSJob a GAMSDatabase is created that is available via the GAMSJob.outDb method. We can use the GAMSDatabase.doExport method to write the content of this database to a gdx file tdata.gdx.

```java
// data from a string with GAMS syntax with explicit export to GDX file
GAMSJob t3 = ws.addJobFromString(getDataText());
t3.run();
t3.outDB().doExport(ws.workingDirectory() + cPathSep + "tdata.gdx");
...
```

### 7.9.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in Transport2 - How to use include files (Transport2).

```java
t3 = ws.addJobFromString(getModelText());
GAMSOptions opt = ws.addOptions();
opt.setDefine("gdxincname", "tdata");
opt.setAllModelTypes("xpress");
t3.run(opt);
...
```

Note that there are some changes in getModelText due to the usage of a GDX file instead of an include file.

```java
if not set gdxincname $abort 'no include file name for data file provided'
@gdxin %gdxincname%
@load i j a b d f
@gdxin
...
```

### 7.9.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs t3a and t3b where the first one contains only the data and the second one contains only the model without data. After running t3a the corresponding outDB can be read in directly just like a gdx file. Note that the database needs to be passed to the GAMSJob.run method as additional argument.

```java
GAMSJob t3a = ws.addJobFromString(getDataText());
GAMSJob t3b = ws.addJobFromString(getModelText());
t3a.run();
opt.setDefine("gdxincname", t3a.outDB().name());
t3b.run(opt, t3a.outDB());
...
```
7.9.3.11 How to define data using C++ data structures (Transport4)

We use the `vector<T>` and `map<Key, Value>` to define C++ data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```cpp
vector<string> plants = {
    "Seattle", "San-Diego"
};
vector<string> markets = {
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = {
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = {
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = {
    { make_tuple("Seattle", "New-York"), 2.5 },
    { make_tuple("Seattle", "Chicago"), 1.7 },
    { make_tuple("Seattle", "Topeka"), 1.8 },
    { make_tuple("San-Diego", "New-York"), 2.5 },
    { make_tuple("San-Diego", "Chicago"), 1.8 },
    { make_tuple("San-Diego", "Topeka"), 1.4 }
};
```

7.9.3.12 How to prepare a GAMSDatabase from C++ data structures (Transport4)

At first we create an empty GAMSDatabase `db` using the `GAMSWorkspace.addDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.addSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C++ data structure and add them by using the `GAMSSet.addRecord` method. We do pretty much the same thing for the parameters.

Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that the scalars can be treated as parameter with dimension 0.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```cpp
GAMSDatabase db = ws.addDatabase();
GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants)
    i.addRecord(p);
GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
    for (string p: plants)
        a.addRecord(p).setValue(capacity[p]);
GAMSParameter d = db.addParameter("d", "distance in thousands of miles", i, j);
    for (auto t : distance)
        d.addRecord(get<0>(t.first), get<1>(t.first)).setValue(t.second);
GAMSParameter f = db.addParameter("f", "freight in dollars per case per thousand miles");
    f.addRecord().setValue(90);
```

7.9.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS checkpoint, the second one uses the `GAMSWorkspace.addJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.run` method. Furthermore, it passes an additional checkpoint argument to the `run` method. That means the GAMSCheckpoint `cp` captures the state of the `GAMSJob`.

```cpp
GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
```
7.9.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `getModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```gams
Scalar bmult demand multiplier /1/
... demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
... Scalar ms 'model status', ss 'solve status';
```

We create a vector with eight different values for this demand multiplier.

```gams
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 }
```

For each entry of the vector we create a GAMSJob `t5` using the `GAMSWorkspace.addJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a GAMSJob combined from the checkpoint plus the content provided by the string.

We run the GAMSJob and echo some interesting data from the `outDB` using the `GAMSDatabase.getParameter` and `GAMSDatabase.getVariable` methods, the `GAMSParameter.findRecord` and `GAMSVariable.findRecord` methods plus the `GAMSParameterRecord.value` and the `GAMSVariableRecord.level` methods.

```gams
for (double b : bmultlist)
{
GAMSJob t5 = ws.addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;", cp);
t5.run();
cout << "Scenario bmult" << b << ":" << endl;
cout << " Modelstatus: " << t5.outDB().getParameter("ms").findRecord().value() << endl;
cout << " Solvestatus: " << t5.outDB().getParameter("ss").findRecord().value() << endl;
cout << " Obj: " << t5.outDB().getVariable("z").findRecord().level() << endl;
}
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.9.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs we already know from `Transport5` in parallel. We create a GAMSCheckpoint `cp` and initialize it by running a GAMSJob. Furthermore we introduce a demand multiplier as we did before.

```gams
... GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 }
...
```

Furthermore, we introduce a new object `ioMutex` that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the vector of demand multipliers we create a thread executing the `runScenario` method.

```gams
// run multiple parallel jobs using the created GAMSCheckpoint
mutex ioMutex;
vector<thread> v;
for (double b : bmultlist)
{v.emplace_back([ws, cp, &ioMutex, b]{runScenario(&ws, cp,&ioMutex,b)});
for (auto& t : v)
t.join();
}```
In function runScenario a GAMSJob is created and run just like in the preceding example of Transport5. The output section is also the same except for the fact that it is locked by using a lock.guard on the ioMutex. That means the threads of runScenario are running in parallel but the output block of different threads cannot be executed in parallel since it is locked using the same ioMutex.

```cpp
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + toString(b) + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat; ", cp);
    t6.run();
    // we need to make the output a critical section to avoid messed up report information
    lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ": " << endl;
    cout << " Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << " Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << " Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
```

While the output in Transport5 is strictly ordered subject to the order of the elements of bmultlist, in Transport6 the output blocks might change their order but the blocks describing one scenario are still appearing together due to the lock.guard mechanism.

If you want a further impression of the impact of the lock.guard, just rerun Transport6 but comment out the lock.guard as follows and compare the output.

```cpp
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + toString(b) + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat; ", cp);
    t6.run();
    // we need to make the output a critical section to avoid messed up report information
    // lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ": " << endl;
    cout << " Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << " Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << " Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
```

7.9.3.16 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In Transport7 the usage of gams::GAMSModelInstance is demonstrated.

At first we create a checkpoint cp as in the preceding examples. Then we create the GAMSModelInstance mi using the GAMSCheckpoint.addModelInstance method. Note that the GAMSJob again contains no solve statement and the demand multiplier is already included with default value 1.

```cpp
GAMSCheckpoint cp = ws.addCheckpoint();
GAMSJob t7 = ws.addJobFromString(getModelText());
t7.run(cp);
GAMSModelInstance mi = cp.addModelInstance();
```

7.9.3.17 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A GAMSModelInstance uses a syncDB to maintain the data. We define bmult as GAMSParameter using the GAMSDatabase.addParameter method and specify cplex as solver. Afterwards the GAMSModelInstance is instantiated with three arguments: the solve statement, the GAMSOptions object opt and bmult. The GAMSModifier means that bmult is modifiable while all other parameters, variables and equations of GAMSModelInstance mi stay unchanged.

We use the GAMSParameter.addRecord method to assign a value to bmult that can be varied afterwards using the GAMSParameter.firstRecord method to reproduce our well-known example with different demand multipliers.
7.9 Tutorial

GAMSModelInstance mi = cp.addModelInstance();
GAMSParameter bmult = mi.syncDb().addParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.addOptions();
opt.setAllModelTypes("cplex");
// instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
mi.instantiate("transport use lp min z", opt, GAMSModifier(bmult));
bmult.addRecord().setValue(1.0);
vector<double> bmultlist = {0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3};
for (double b : bmultlist) {
bmult.firstRecord().setValue(b);
mi.solve();
cout << "Scenario bmult=" << b  << ":"  << endl;
cout << " Modelstatus: "  << mi.modelStatusAsString()  << endl;
cout << " Solvestatus: "  << mi.solveStatusAsString()  << endl;
cout << " Obj: "  << mi.syncDb().getVariable("z").findRecord().level()  << endl;
}

7.9.3.18 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a GAMSModelInstance and define \( x \) as GAMSVariable and its upper bound as GAMSParameter \( xup \). At the following instantiate method GAMSModifier has three arguments. The first one says that \( x \) is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GAMSParameter that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

GAMSVariable x = mi.syncDb().addVariable("x", 2, GAMSEnum::VarType::Positive, "");
GAMSParameter xup = mi.syncDb().addParameter("xup", 2, "upper bound on x");
// instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare upper bound of X mutable
mi.instantiate("transport use lp min z", GAMSModifier(x, GAMSEnum::SymbolUpdateAction::Upper, xup));
for (GAMSSetRecord i : t7.outDB().getSet("i")) {
  for (GAMSSetRecord j : t7.outDB().getSet("j")) {
xup.clear();
xup.addRecord(i.key(0), j.key(0)).setValue(0);
mi.solve();
cout << "Scenario link blocked"  << i.key(0)  << "-"  << j.key(0)  << endl;
cout << " Modelstatus: "  << mi.modelStatusAsString()  << endl;
cout << " Solvestatus: "  << mi.solveStatusAsString()  << endl;
cout << " Obj: "  << mi.syncDb().getVariable("z").findRecord().level()  << endl;
}
}

7.9.3.19 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint \( cp \) from a GAMSJob. Then we define a vector that represents the different values of the demand multiplier. While ioMutex is used to avoid messed up output, vectorMutex synchronizes the access to the vector. Then we start multiple thread executing the method scenSolve. The number of parallel executed threads is specified by nrThreads. All threads share the same vector that provides the different values for \( bmult \).

GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
vector<double> bmultVector = {1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6};
int nrThreads = 2;
std::mutex vectorMutex;
std::mutex ioMutex;
vector<thread> v;
for (int i = 0; i < nrThreads; i++) {
v.emplace_back([ws, &cp, &bmultVector, &vectorMutex, &ioMutex] {scenSolve(&ws, &cp, &bmultVector, &vectorMutex, &ioMutex);});
for (auto& t : v) t.join();
}
In `scenSolve` we create and instantiate a `GAMSModelInstance` as in the preceding examples and make `bmult` modifiable. As long as `bmultVector` is not empty, an elements is taken and removed from the `vector`. The value is used for the scalar `bmult`. Note that we chose `cplex` as solver because it is thread safe (gurobi would also be possible). Once the `vector` is empty the loop terminates.

```cpp
void Transport8::scenSolve(GAMSWorkspace* ws, GAMSCheckpoint* cp, vector<double>* bmultVector, std::mutex* vectorMutex, std::mutex* ioMutex)
{
    unique_lock<mutex> vectorLock(*vectorMutex);
    GAMSModelInstance mi = cp->addModelInstance();
    vectorLock.unlock();
    GAMSParameter bmult = mi.syncDb().addParameter("bmult", "demand multiplier");
    GAMSOptions opt = ws->addOptions();
    opt.setAllModelTypes("cplex");
    // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
    mi.instantiate("transport use lp min z", opt, GAMSModifier(bmult));
    bmult.addRecord().setValue(1.0);
    while (true)
    {
        double b;
        // dynamically get a bmult value from the vector instead of passing it to the different threads at creation time
        vectorLock.lock();
        if (bmultVector->empty())
            return;
        b = bmultVector->back();
        bmultVector->pop_back();
        vectorLock.unlock();
        bmult.firstRecord().setValue(b);
        mi.solve();
        // we need to make the output a critical section to avoid messed up report informations
        unique_lock<mutex> ioLock(*ioMutex);
        cout << "Scenario bmult=" << b << ":" << endl;
        cout << " Modelstatus: " << mi.modelStatus() << endl;
        cout << " Solvestatus: " << mi.solveStatus() << endl;
        cout << " Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
        ioLock.unlock();
    }
}
```

7.9.3.20 How to fill a `GAMSDatabase` by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a `GAMSDatabase`. It can only be run on Windows since it makes use of the Microsoft Access Driver. Furthermore the example makes use of the Qt SQL module. Note that you need to have Microsoft Access installed and that its bitness needs to match the bitness of your GAMS version. In the example we call a function `readFromAccess` that finally returns a `GAMSDatabase` as shown below.

```cpp
GAMSDatabase db = readFromAccess(ws);
```

The data we are going to read can be found in `<GAMS system directory>\apifiles\Data\transport.accdb`. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty `GAMSDatabase`. Afterwards we create a `QSqlDatabase`, specify the connection string and open the connection to the database. If opening the connection was successful, data is read from the database into the `GAMSDatabase` using the methods `readSet` and `readParameter`.

```cpp
GAMSDatabase readFromAccess(GAMSWorkspace ws)
{
    QCoreApplication::instance()->removeOrganizationName("GAMS");
    QFile file(ws.systemDirectory() + QDir::separator() + "apifiles\Data\transport.accdb");
    if (!file.open(QIODevice::ReadOnly))
        return false;

    QSqlDatabase db = QSqlDatabase::addDatabase("QODBC", ws->organizationName());
    db.setDatabaseName(file.fileName());
    if (!db.open())
    {
        cout << "Error: Failed to create a database connection. " << db.lastError().text().toStdString() << endl;
        exit(1);
    }

    // read GAMS sets
    readSet(db, SELECT Plant FROM Plant, "i", 1, "canning plants");
    readSet(db, SELECT Market FROM Market, "j", 1, "markets");

    // read GAMS parameters
    readParameter(db, SELECT Plant, Capacity FROM Plant, "a", 1, "capacity of plant i in cases");
    readParameter(db, SELECT Market, Demand FROM Market, "b", 1, "demand at market j in cases");
    readParameter(db, SELECT Plant, Market, Distance FROM Distance, "d", 2, "distance in thousands of miles");
    sqlDb.close();
}
```
The function `readSet` creates a `QSqlQuery` that reads data from a table of the Access database. A `GAMSSet` is created and populated with the data.

```cpp
void readSet(QSqlDatabase sqlDb, GAMSDatabase db, string strAccessSelect, string setName, int setDim, string setExp = "")
{
    QSqlQuery query(sqlDb);
    if (!query.exec(strAccessSelect.c_str()))
    {
        cout << "Error executing query on set '" << setName << "'" << endl;
        cout << query.lastError().text().toStdString() << endl;
        exit(1);
    }
    if (query.size() && (query.record().count() != setDim))
    {
        cout << "Number of fields in select statement does not match setDim" << endl;
        exit(1);
    }
    GAMSSet i = db.addSet(setName, setDim, setExp);
    vector<string> keys = vector<string>(setDim);
    while (query.next())
    {
        for (int idx = 0; idx < setDim; idx++)
            keys[idx] = query.value(idx).toString().toStdString();
        i.addRecord(keys);
    }
}
```

Once we read in all the data we can create a `GAMSJob` from the `GAMSDatabase` and run it as usual. Finally the results are written to `transport.accdb`.

### 7.9.3.21 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `<GAMS system directory>/apifiles/Data\transport.xlsx`. It can only be run on Windows. Note that you need to have Microsoft Excel installed and that its bitness needs to match the bitness of your GAMS version. The model is given as string without data like in in many examples before. At first we have to add

```cpp
#include <QAxObject>
#include <Windows.h>
```

to be able to use the `QAxObject` class, which serves as a wrapper for COM objects. Furthermore we include `Windows.h` to access the functions `CoInitialize` and `CoUninitialize`. In the main function we open the aforementioned file to get access to its content. Note that we need to call `CoInitialize(0)` first in order to initialize ActiveX. We create an instance of the class `QAxObject` and use the `querySubObject` method multiple times to get to the sheets of the workbook.

```cpp
QAxObject* excel = new QAxObject( "Excel.Application", 0 );
QAxObject* workbooks = excel->querySubObject( "Workbooks" );
QAxObject* workbook = workbooks->querySubObject( "Open(const QString&)", fileName );
QAxObject* sheets = workbook->querySubObject( "Worksheets" );
...```

The method `sheetToParameter` is used in order to transfer data from the workbook into `GAMSDParameter` instances. In order to determine from which sheet the actual data needs to be read, the method takes the `QAxObject` instance referring to the sheets and the actual sheet name. A new `GAMSDParameter` is added to the `GAMSDatabase` object `db` using `set1` and `set2` as domains. The next step is to determine the number of columns and the number of rows used. The `for` loop reads the actual data, adds the records to the new `GAMSDParameter` and sets their values.

```cpp
GAMSDParameter sheetToParameter(QAxObject* sheets, string sheetName, GAMSDatabase db, string paramName, string paramText, GAMSSet set1, GAMSSet set2)
{
    QAxObject* sheet = sheets->querySubObject( "Item( string )", sheetName.c_str() );
    vector<GAMSDomain> sets [set1, set2];
    GAMSDParameter param = db.addParameter(paramName, paramText, sets);
    QAxObject* columns = sheet->querySubObject("Columns");
    int intCols = columns->property("Count").toInt();
    QAxObject* rows = sheet->querySubObject("Rows");
    int intRows = rows->property("Count").toInt();
...```
for (int j = 2; j <= intCols; j++) {
    string namej = sheet->querySubObject("Cells( int, int )", 1, j)->dynamicCall("Value()").toString().toStdString();
    for (int i = 2; i <= intRows; ++i) {
        string namei = sheet->querySubObject("Cells( int, int )", i, 1)->dynamicCall("Value()").toString().toStdString();
        GAMSParameterRecord rec = param.addRecord(namei, namej);
        double value = sheet->querySubObject("Cells( int, int )", i, j)->dynamicCall("Value()").toDouble();
        rec.setValue(value);
    }
}
return param;
}

After all required data was read from the workbook, we need to close it and quit the Excel application.
... 
workbook->dynamicCall("Close()");
excel->dynamicCall("Quit()");
...

Now we can create und run the GAMSJob using the created GAMSDatabase as usual.

GAMSOptions opt = ws.addOptions();
GAMSJob t10 = ws.addJobFromString(getModelText());
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t10.run(opt, db);

Finally we need to call ‘CoUninitialize’ in order to close the COM library.
... 
::CoUninitialize();
...

7.9.3.22 How to create and use a save/restart file (Transport11)

In Transport11 we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a GAMSCheckpoint. We start by creating the save/restart file by calling createSaveRestart.

... 
std::string cpName = "tbase";
createSaveRestart(argc, argv, cpName);
...

In function createSaveRestart we choose the checkpointName as relative path to the current directory in order to create a GAMSWorkspace. Then we create a GAMSJob from a string. Note that the string given via getBaseModelText() contains the basic definitions of sets without giving them a content (that is what $onempty is used for). Afterwards we specify a GAMSOption to only compile the job without executing it. Then we create a checkpoint cp that is initialized by the following run of the GAMSJob and stored in the file given as argument to the function. This becomes possible because the addCheckpoint method accepts identifiers as well as file names as argument.

```c++
void createSaveRestart(int argc, char* argv[], const string &checkpointName) {
    GAMSWorkspaceInfo wsInfo;
    if (argc > 1) 
        wsInfo.setSystemDirectory(argv[1]);
    wsInfo.setWorkingDirectory(".", (char*)checkpointName);
    GAMSWorkspace ws(wsInfo);
    GAMSJob j1 = ws.addJobFromString(getBaseModelText());
    GAMSOptions opt = ws.addOptions();
    opt.setAction(GAMSOptions::EAction::CompileOnly);
    auto checkpoint = ws.workingDirectory() + checkpointName;
    GAMSCheckpoint cp = ws.addCheckpoint(checkpoint);
    j1.run(opt, cp);
}
```

So what you should keep in mind before we return to further explanations of the main function is, that now the file tbase in folder tbase contains a checkpoint. Now in the main function we define some data using C++ data structures as we already did in Transport4 before we create another GAMSWorkspace.
Afterwards we set up the GAMSDatabase like we already did in Transport4. Once this is done we run a GAMSJob using this data plus the checkpoint stored in file `tbase`.

```cpp
GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");
GAMSOptions opt = ws.addOptions();
GAMSJob t4 = ws.addJobFromString(getModelText(), cpBase);
opt.setAllModelTypes("xpress");
t4.run(opt, db);
```

Note that the string from which we create job `t4` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the GAMSDatabase correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

### 7.9.4 How to use the pre configured example projects

Depending on the operating system, the GAMS C++ API comes with example project configurations for different development tools like CMake, qmake and Visual Studio. They provide an easy way of building and running the distributed GAMS C++ examples and are located in `<GAMS system directory>\apifiles\C++`.

#### 7.9.4.1 CMake

**Windows:**

Create build directory

```bash
cd <GAMS system directory>\apifiles\C++
mkdir build && cd build
```

Run CMake and build project

```bash
cmake -G "Visual Studio 16 2019" -config Release -DVSVERSION:STRING=vs2019 ..
msbuild.exe examples.sln /p:Configuration=Release
```

Note

For other Visual Studio versions, the generator-name (-G) needs to be adjusted (see cmake documentation).

All generated executables can now be found in the CMake build directory. The exact path depends on the project name and the config setting. Assuming the solution was build with `-config Release`, a `transport1.exe` executable is located in `<GAMS system directory>\apifiles\C++\build\transport1\Release`. Therefore you can use the following commands to execute the `transport1` example: In order to execute an example (e.g. `transport1`) run the following commands:
cd transport1\Release
set PATH=%PATH%;<GAMS system directory>\apifiles\C++\lib\vs2019
transport1.exe

Linux:

Create build directory

cd <GAMS system directory>/apifiles/C++
mkdir build && cd build

Run CMake and build project

cmake ..
make

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

cd transport1
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1

Mac OS X:

Create build directory

cd <GAMS system directory>/apifiles/C++
mkdir build && cd build

Run CMake and build project

cmake ..
make

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

cd transport1
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1
7.9 Tutorial

7.9.4.2 qmake

Building the examples can be done by using Qt Creator. Open `<GAMS system directory>\apifiles\C++\examples.pro` with Qt Creator. After the project is loaded, click on Build > Build All in order to compile and link the project. After the build process has carried out, select a subproject and click on Build > Run in order to execute the chosen example.

Alternatively you can run qmake using the command line.

**Windows:**

Create build directory

```bash
cd <GAMS system directory>\apifiles\C++
mkdir build && cd build
```

Run qmake and build project

```bash
qmake ..\examples.pro VSVERSION=vs2019
nmake
```

Note

For other supported Visual Studio versions change the value of `VSVERSION`.

All generated executables can now be found in the qmake output directory. In order to execute an example (e.g. transport1) run the following commands:

```bash
cd ..\bin
set PATH=%PATH%;<GAMS system directory>\apifiles\C++\lib\vs2019
transport1.exe
```

**Linux and Mac OS X:**

In order to run qmake on Linux and Mac OS X refer to the description of using CMake above and call `qmake` instead of `cmake`.

7.9.4.3 Microsoft Visual Studio

The Visual Studio solutions are available for Windows only. Open the solution file that matches the version of your Visual Studio installation (e.g. `<GAMS system directory>\apifiles\C++\examples-vs2017.sln`).

As soon as the solution is loaded, click on Build > Configuration Manager and adjust the Active solution configuration and the Active solution platform. The solution configuration needs to be set to Release for using the distributed binaries of the GAMS C++ API. Note that the platform needs to match the version of your GAMS installation.

The examples Transport9 and Transport10 require a Qt installation. If you do not have one, you need to disable these projects. Right click on a project in the Solution Explorer and choose Unload Project from the context menu.

Click on Build > Build Solution in order to compile and link the examples.

Note

In some cases the build might return with `xcopy` errors. Just build the project again. Make sure that you don’t do a rebuild.

In order to run an example, click on Debug > Start Without Debugging. This will execute the current StartUp project. Changing the StartUp project can be achieved by right clicking on a project in the Solution Explorer and choosing Set as StartUp Project.
7.10 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Java API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known transportation problem. These examples introduce several API features step by step.

- **Getting started** A quick introduction about how to compile and run a program from command line or an IDE
- **Important Classes of the API** Overview of some fundamental classes of the GAMS Java API
- **How to use API** An extensive set of examples describing how to use API components

7.10.1 Getting started

A Java program that uses GAMS Java API requires at least Java SE 7 (Since 25.0) to compile and run. For all platforms, assume GAMS system has been installed at `[PathToGAMS]` directory, called GAMS directory. The directory `[PathToGAMS]` denotes the path setting according to your GAMS installation on targeted platforms. For instance

- on Windows-based platforms with GAMS distribution 40.1 installed, `[PathToGAMS]` denotes `C:\GAMS\win64\40.1`
- on Unix-based platforms with GAMS distribution 40.1 installed, `[PathToGAMS]` denotes `/usr/gams/gams40.1_linux_x64_64.sfx`

All GAMS Java API classes are contained within one single jar file `GAMSJavaAPI.jar` with a namespace `com.gams.api`. The jar file is distributed with the current GAMS distribution and located at

- on Windows-based platforms: `[PathToGAMS]`\apifiles\Java\api\GAMSJavaAPI.jar
- on Unix-based platforms: `[PathToGAMS]`/apifiles/Java/api/GAMSJavaAPI.jar

There are two different approaches of how to use the GAMS Java API. Either the Java source files can be edited with any common editor and compiled from command line or a Java IDE can be used. The following sections give a quick introduction about the different possibilities.

7.10.1.1 Compiling a Program

To compile a Java program, one usually provides the following information to Java compiler:

- the directory(ies) containing all required Java class files
- the directory to place the generated class files
- the name of the Java source file

A Java program that uses GAMS Java API requires class files that are contained in `GAMSJavaAPI.jar` for compilation.
7.10 Tutorial

7.10.1.1 Compiling a Program from Command Line  To compile a Java program that uses GAMS Java API, for instance HelloAPI.java, at the command line:

```
javac -cp [CLASSPATH] -d [TARGETDIR] HelloAPI.java
```

where

[CLASSPATH] is 'GAMSJavaAPI.jar' with its location, as it contains classes files that are used by

- On Windows-based platform, [CLASSPATH] denotes [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar
- On Unix-based platforms, [CLASSPATH] denotes [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR] is the directory to place the generated classe file.

Please note that it is possible to add [CLASSESPATH] to your CLASSPATH environment variable of the operating system as an alternative to specifying -cp [CLASSESPATH]. Also note that -d [TARGETDIR] is optional. In case -d [TARGETDIR] is omitted and the compilation is successful, the class file will be generated and located under the current directory.

To compile other programs, change the arguments accordingly. See also How to compile and run examples from the GAMS system directory.

7.10.1.2 Compiling a Program from Java IDE  In case of compiling a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), the location of the jar file GAMSJavaAPI.jar shall be added into the Java build path of the IDE's project properties.

For an Eclipse user, see How to import an Eclipse Java Project from the GAMS system directory.

7.10.1.2 Running a Program

To run a Java program, one usually provide the following information to Java Virtual Machine:

- the directory(ies) containing all required Java classes
- the directory(ies) containing all required shared libraries
- the name of an entry point class (in most case containing main method)

A Java program that uses GAMS Java API requires a number of shared libraries for establishing a connection with GAMS software components during run time. These shared libraries are platform dependent and they are located at [PathToGAMS]/apifiles/Java/api directory.
7.10.1.2.1 Running a Program from Command Line  To run a Java program that uses GAMS Java API, for instance HelloAPI.class containing in [TARGETDIR] directory, at the command line type:

```
java -cp [CLASSPATH] HelloAPI
```

where

[CLASSPATH] is the list of two paths, a path of 'GAMSJavaAPI.jar' and a path of targeted directory containing 'HelloAPI.class', separated by path separator on the targeted platform. On Windows-based platform, [CLASSESPATHS] denotes [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;[TARGETDIR] On Unix-based platform, [CLASSESPATHS] denotes [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar:[TARGETDIR]

HelloAPI is the name of an entry point class containing main method

Please note that it is possible to add [CLASSPATH] to your CLASSPATH environment variable of the operating system as an alternative to specify -cp [CLASSPATH].

Most operations performed by GAMS Java API such as gdx or options operations require shared libraries (located in the folder [PathToGAMS]/apifiles/Java/api by default) when running a program. In case no java.library.path has been specified when running a program, the Java API will determine the path of the shared libraries from the java class path containing GAMSJavaAPI.jar.

Note that it is also possible to pass the following argument to Java Virtual Machine:

```
-Djava.library.path=[LIBRARYPATH]
```

where [LIBRARYPATH] describes the list of paths containing all required shared libraries (as previously mentioned, it is [PathToGAMS]/apifiles/Java/api by default). In such case, the Java API will give priority to the java library path when loading all required JNI libraries during run time. The java API will raise an exception when either the java virtual machine fails to load the required libraries from the specified java library path or there is a version conflict of shared libraries during run time.

To run other programs, change the arguments accordingly. See also How to compile and run examples from the GAMS system directory.

7.10.1.2.2 Running a Program from Java IDE  In case of running a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), it is possible to set the necessary arguments via the properties of the IDE project.

For an Eclipse user, see How to import an Eclipse Java Project from the GAMS system directory.
7.10 Tutorial

7.10.1.3 Setting Up Your Environment

Since 24.3, the GAMS system directory can be specified within the program during run time. Therefore the setting up of execution environment before running a program is no longer required (see also Release Notes 24.3).

Nevertheless, it is still possible to preconfigure the GAMS system directory before running a program and use the default workspace setting within a program. In the default setting, GAMS system directory is not specified by a user within the program. The directory will be searched during run time from environment variable in the following order (depends on the target platform):

- On Windows-based platform: first from PATH environment variable. If not found, from the platform windows registry gams.location,
- On Mac-based platform: first from PATH environment variable. If not found, from DYLD_LIBRARY_PATH,
- On other Unix-based platform: from PATH environment variable. If not found, from LD_LIBRARY_PATH.

The following examples illustrate how to add [PathToGAMS] into the environment variable PATH on different platforms:

- On Windows 2000, XP, Vista, or Windows 7 desktop:
  > right-click on 'My Computer'
  > choose 'Properties' (alternatively, click on 'System' icon in the control panel)
  > click on 'Advanced' tab (or 'Advance system settings')
  > click on 'Environment Variables'
  > edit 'PATH' by adding [PathToGAMS] to the variable using a semi-colon as a separator.

- On Window-based platform via command line terminal:
  set PATH=[PathToGAMS];%PATH%

- On Unix-based platform via command line terminal using Bourne shell and its derivatives:
  export PATH=[PathToGAMS]:${PATH}

- On Unix-based platform via command line terminal using C Shell:
  setenv PATH [PathToGAMS]:${PATH}

7.10.1.4 How to compile and run examples from the GAMS system directory

GAMS provides several examples to demonstrate how to use GAMS Java API. These examples are located in the [PathToGAMS]/apifiles/Java directory containing various examples based on the different optimization problems. The following explanations guide you through the compilation and execution process of one example based on the well-known transportation problem, assuming that Java Runtime System is already installed on your machine.

For instance, Transport1.java under the directory [PathToGAMS]/apifiles/Java/transport demonstrates how to retrieve GAMS transport model from GAMS Model Library, execute the model with various GAMS options, and extract results after execution.

To compile Transport1.java at the command line under the directory [PathToGAMS]/apifiles/Java/transport:
javac -cp [CLASSPATH] -d [TARGETDIR] Transport1.java

where

[CLASSPATH] is a location of 'GAMSJavaAPI.jar', as it contains classes files that are used by 'Transport1.java'.
On Windows-based platform, [CLASSPATH] denotes
[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;
On Unix-based platforms, [CLASSESPATHS] denotes
[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR] is the destination directory to place the generated classed file.

As Transport1.java is declared under package com.gams.examples.transport, the output class file will be located in [TARGETDIR] under the directory structure corresponding to package information. This means the compiled output file Transport1.class is located under:

• [TARGETDIR]\com\gams\examples\transport on Windows-based platform, or
• [TARGETDIR]/com/gams/examples/transport on Unix-based platforms.

To run Transport1.class from the command line:

java -cp [CLASSPATH] com.gams.examples.transport.Transport1

where

[CLASSPATH] is the list of two paths, 'GAMSJavaAPI.jar' with its path and a directory containing 'Transport1.class' and (in this case [TARGETDIR] from the compilation step), separated by path separator on the targeted platform.
On Windows-based platform, [CLASSESPATHS] denotes
[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;[TARGETDIR]
On Unix-based platforms, [CLASSESPATHS] denotes
[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar:[TARGETDIR]

Please note that the path [TARGETDIR] can be either absolute or relative path representing the directory that contains Transport1.class.

To compile and run other examples under the directory [PathToGAMS]/apifiles/Java, adjust the arguments accordingly.

See How to use API for detailed explanations on the series of transportation problem examples located in the folder [PathToGAMS]/apifiles/Java/transport.
7.10 Tutorial

7.10.1.5 How to import an Eclipse Java Project from the GAMS system directory

Since GAMS version 24.2 there is an prepared Java project that can be imported into Eclipse. The project is located in folder [PathToGAMS]/apifiles/Java/Eclipse and contains the examples based on the transportation problem. The following explanations guide you through the import and preparation process, assuming that the Eclipse Java IDE is already installed on your machine.

To import the project located in folder [PathToGAMS]/apifiles/Java/Eclipse into the workspace:

1. open the Eclipse IDE with the chosen workspace location, click on File menu and choose Import.
2. In the Import window, choose General > Existing Projects into Workspace then click Next.
3. In the next page of Import window, click Browse.. and select the folder with the prepared project ([PathToGAMS]/apifiles/Java/Eclipse) as root directory.
   We recommend to check for option Copy projects into workspace. Click Finish to finish the Import window.
4. Now the project is imported, and appears in the Package Explorer window on the left hand side of of the IDE (shown below with its elements expanded).
5. As every example in the project requires GAMS Java API classes files to compile (all classes are packaged in GAMSJavaAPI.jar located in the folder [PathToGAMS]/apifiles/Java/api), you need to tell eclipse where to find GAMSJavaAPI.jar. To do this, either click on the project then choose Project > Properties or right click on the project name and choose Properties.
6. The Properties window of the project appears. Click Java Build Path on the left of the window, choose Libraries tab, choose Add External JARs,... select GAMSJavaAPI.jar located in the folder [PathToGAMS]/apifiles/Java/api to be opened. and click OK to finish adding [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar to Java Build Path.
7. Now all the java files should be successfully compiled without errors except for Transport10 (unsuccesful compilation is denoted with red x mark in front of the file name). This is because Transport10 requires an additional jexcelapi JAR file jxl.jar from jexcelapi. It can be downloaded and unzipped the downloaded archive into a local directory e.g. C:\tools. Then the jar file C:\tools\jexcelapi\jxl.jar can be added to the the Java Build Path of the project in a similar way as explained in 6.
8. To run a transport example, for example Transport1, you need to create a Run Configuration.
   To do this, either click Run menu and choose Run Configurations... or open the drop down menu next to the run button and click on Run Configurations.... Then create a new run configuration by either press the New button in the left hand side of the window or right click at Java Application in the left hand side of the window and choose New.
9. The run configuration for Transport1 appears.
   For each run configuration it is possible to configure the execution of a program, such as the arguments of the virtual machine, class paths, environment variables and so on.
   For instance, to specify the java.library.path for Transport1 run configuration, choose Arguments tab in the run configuration and specify -Djava.library.path=[PathToGAMS]/apifiles/Java/api as one of the VM arguments. Then Click Apply and Run.

Note that the run configuration described in 8. and 9. is needed to be set for every single example. Once this is done you should be able to use the run configuration to repeat the execution of all the transport examples.

For a more detailed explanations on each transport examples see How to use API.
7.10.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples. All GAMS Java API classes are contained within one single jar file GAMSJavaAPI.jar with a namespace com.gams.api. It provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this namespace allow convenient exchange of input data and model results (GAMSDatabase) and help to create and run GAMS models (GAMSJob), that can be customized by GAMS options (GAMSOptins). Furthermore, it introduces a way to solve a sequence of closely related models in the most efficient way (GAMSModelInstance).

Other classes are GAMSWorkspace, GAMSOptins, GAMSSymbol, and GAMSException.

7.10.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the Java API. [PathToGAMS]/apifiles/Java/transport contains multiple examples dealing with the well-known transportation problem. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)
- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to retrieve a solution from an output database (Transport1)
- How to specify the solver using GAMSOptins (Transport1)
- How to run a job with a solver option file and capture its log output (Transport1)
- How to use include files (Transport2)
- How to set a non-default working directory (Transport3)
- How to read data from string and export to GDX (Transport3)
- How to run a job using data from GDX (Transport3)
- How to run a job using implicit database communication (Transport3)
- How to define data using Java data structures (Transport4)
- How to prepare a GAMSDatabase from Java data structures (Transport4)
- How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)
- How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)
- How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)
- How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)
- How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)
- How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)
- How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)
- How to fill a GAMSDatabase by reading from MS Access (Transport9)
- How to fill a GAMSDatabase by reading from MS Excel (Transport10)
- How to create and use a save/restart file (Transport11)
7.10 Tutorial

7.10.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing `Transport1` with `C:/GAMS/win64/40.1` we use the 64-bit version of GAMS 40.1 to run `Transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```java
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
if (args.length > 0)
    wsInfo.setSystemDirectory( args[0] );
GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
```

Remember that the bitness of the GAMS system has to match the bitness of your Java Runtime Environment.

7.10.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented Java API offers much more than exchanging data between Java and GDX, a common use case is the export and import of GDX files. The central class for this purpose is `GAMSDatabase`. We assume that the data to be exported is available in Java data structures.

```java
List<String> plants = Arrays.asList("Seattle", "San-Diego");
List<String> markets = Arrays.asList("New-York", "Chicago", "Topeka");
Map<String, Double> capacity = new HashMap<String, Double>()
    .put("Seattle", Double.valueOf(350.0));
    .put("San-Diego", Double.valueOf(600.0));
Map<String, Double> demand = new HashMap<String, Double>
    .put("New-York", Double.valueOf(325.0));
    .put("Chicago", Double.valueOf(300.0));
    .put("Topeka", Double.valueOf(275.0));
Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>
    .put( new Vector<String>(Arrays.asList("Seattle", "New-York") ), Double.valueOf(2.5));
    .put( new Vector<String>(Arrays.asList("Seattle", "Chicago") ), Double.valueOf(1.7));
    .put( new Vector<String>(Arrays.asList("Seattle", "Topeka") ), Double.valueOf(1.8));
    .put( new Vector<String>(Arrays.asList("San-Diego", "New-York") ), Double.valueOf(2.5));
    .put( new Vector<String>(Arrays.asList("San-Diego", "Chicago") ), Double.valueOf(1.8));
    .put( new Vector<String>(Arrays.asList("San-Diego", "Topeka") ), Double.valueOf(1.4));
```

Different type of GAMS symbols are represented using different Java data structures. The data for the GAMS sets is represented using `List` of Strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by `Map` (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses `Vector`s for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with Java data structures.

A new `GAMSDatabase` instance can be created using `GAMSWorkspace.addDatabase`.

```java
// create new GAMSDatabase instance
GAMSDatabase db = ws.addDatabase();
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.addRecord`.

```java
GAMSSet i = db.addSet("i", 1, "canning plants");
for(String p : plants)
    i.addRecord(p);
```
GAMSParameter instances can be added by using the method GAMSDatabase.addParameter. In this example we use the overloaded method which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

```java
GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (String p : plants)
    a.addRecord(p).setValue(capacity.get(p));
```

As soon as all data is prepared in the GAMSDatabase, the method GAMSDatabase.export can be used to create a GDX file.

```java
db.export("data.gdx");
```

### 7.10.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using GAMSWorkspace.addDatabaseFromGDX. The method takes a path to a GDX file and creates a GAMSDatabase instance.

```java
GAMSDatabase gdxdb = ws.addDatabaseFromGDX("data.gdx");
```

Reading the data from the GAMSSet $i$ into a List of Strings can be done as follows:

```java
List<String> gdxPlants = new ArrayList<String>();
for(GAMSSetRecord rec : gdxdb.getSet("i"))
    gdxPlants.add(rec.getKey(0));
```

A new List `gdxPlants` is created. $i$ is retrieved by calling GAMSDatabase.getSet on `gdxdb`. The returned GAMSSet object can be iterated using a for-loop to access the records of the set. Each record is of type GAMSSetRecord and can be asked for its keys.

You can do the same for GAMSParameter. Instead of creating a List, we want to have the data in the form of a Map. GAMSParameterRecord can not only be asked for its keys, but also for its value. The following code snippet shows how to read the one dimensional parameter $a$ into a Map `<String, Double>`.

```java
Map<String, Double> gdxCapacity = new HashMap<String, Double>();
for(GAMSParameterRecord rec : gdxdb.getParameter("a"))
    gdxCapacity.put(rec.getKey(0), rec.getValue());
```

For a key of multi dimensional symbol, we choose Vector of String instead of String.

```java
Map<Vector<String>, Double> gdxDistance = new HashMap<Vector<String>, Double>();
for(GAMSParameterRecord rec : gdxdb.getParameter("d"))
    gdxDistance.put( new Vector<String>(Arrays.asList(new String[] {rec.getKey(0), rec.getKey(1)})), rec.getValue());
```

Scalar can be read into a variable of type double by accessing the value of the first and only record.

```java
double gdxFreight = gdxdb.getParameter("f").getFirstRecord().getValue();
```

### 7.10.3.4 How to run a GAMSJob from file (Transport1)

At first we create our workspace using GAMSWorkspace ws = new GAMSWorkspace();. Afterward we can create a GAMSJob `t1` using the addJobFromGamsLib method and run it.

Apparently you can create a GAMSJob with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GAMSJob `t1` can be defined using the GAMSJob.addJobFromFile method.

```java
// create GAMSWorkspace "ws" with default working directory
// (the directory named with current date and time under System.getProperty("java.io.tmpdir"))
GAMSWorkspace ws = new GAMSWorkspace();
// create GAMSJob "t1" from "transport" model in GAMS Model Libraries
GAMSJob t1 = ws.addJobFromGamsLib("transport");
// run GAMSJob "t1"
t1.run();
```

See also Transport1.java.
7.10 Tutorial

7.10.3.5 How to retrieve a solution from an output database (Transport1)

The following lines create the solution output and illustrate the usage of the `GAMSJob.OutDB` property to get access to the `GAMSDatabase` created by the `run` method. To retrieve the content of variable `x` we use the `getVariable` method and the `GAMSVariableRecord` class.

```java
... // retrieve GAMSVariable "x" from GAMSJob's output databases
System.out.println("Ran with Default: ");
GAMSVariable x = t1.OutDB().getVariable("x");
for (GAMSVariableRecord rec : x)
{
    System.out.print("x( + rec.getKeys()[0] + ", " + rec.getKeys()[1] + ") : ");
    System.out.print(" + rec.getLevel());
    System.out.println(" + rec.getMarginal()");
}
... See also Transport1.java.
```

7.10.3.6 How to specify the solver using `GAMSOptions` (Transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.addOptions` method. The `GAMSOptions.setAllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle. Then we run our `GAMSJob t1` with the new `GAMSOptions`.

```java
... // create GAMSOptions "opt1"
GAMSOptions opt1 = ws.addOptions();
// set all model types of "opt1" for "xpress"
opt1.setAllModelTypes("xpress");
// run GAMSJob "t1" with GAMSOptions "opt1"
t1.run(opt1);
... See also Transport1.java.
```

7.10.3.7 How to run a job with a solver option file and capture its log output (Transport1)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterward we use a `GAMSOptions` just like in the preceding example and `GAMSOptions.setOptFile` property to 1 to tell the solver to look for a solver option file. We specify the argument `output` in order to stream the log of the GAMSJob into the file `transport1.xpress.log`. When the `output` argument is omitted then the log will be written to standard output.

```java
... // write file "xpress.opt" under GAMSWorkspace's working directory
try {
    BufferedWriter optFile = new BufferedWriter(new FileWriter(
        ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "xpress.opt");
    optFile.write("algorithm=barrier");
    optFile.close();
} catch (IOException e) {
    e.printStackTrace();
    System.exit(-1);
}
// create GAMSOptions "opt2"
GAMSOptions opt2 = ws.addOptions();
// set all model types of "opt2" for "xpress"
opt2.setAllModelTypes("xpress");
// for "opt2", use "xpress.opt" as solver's option file
opt2.setOptFile(1);
try {
    // run GAMSJob "t2" with GAMSOptions "opt2" and capture log into "transport1.xpress.log".
    PrintStream output = new PrintStream(new File(ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "transport1.xpress.log");
    t1.run(opt2, output);
} catch (FileNotFoundException e) {
    t1.run(opt2);
}
... See also Transport1.java.
```
7.10.3.8 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings data and model are using GAMS syntax.

At first we write an include file \texttt{tdata.gms} that contains the data but not the model text:

\begin{verbatim}
try {
    BufferedWriter file = new BufferedWriter(new FileWriter(ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gms" ));
    file.write(data);
    file.close();
} catch (IOException e) {
    e.printStackTrace();
    System.exit(-1);
}
\end{verbatim}

Afterwards we create a \texttt{GAMSJob} using the \texttt{GAMSWorkspace.addJobFromString} method. \texttt{GAMSOptOptions.defines} is used like the the 'double dash' GAMS parameters, i.e. it corresponds to \texttt{--incname=tdata} on the command line where incname is used as name for the include file in the \texttt{model} string.

\begin{verbatim}
// create GAMSJob "t2" from the "model"
GAMSJob t2 = ws.addJobFromString(model);
// create GAMSOptOption "opt" and define "incname" as "tdata"
GAMSOptOptions opt = ws.addOptions();
opt.defines("incname", "tdata");
// run GAMSJob "t2" with GAMSOptOptions "opt"
t2.run(opt);
\end{verbatim}

The string model contains the following lines to read in the data.

\begin{verbatim}
$if not set incname $abort 'no include file name for data file provided'
$include \$incname
\end{verbatim}

See also \texttt{Transport2.java}.

7.10.3.9 How to set a non-default working directory (Transport3)

At first we create a new directory. Once this is done we can use this directory when creating the \texttt{GAMSWorkspace} and make it the working directory.

\begin{verbatim}
// create a directory
File workingDirectory = new File(System.getProperty("user.dir"), "Transport3");
workingDirectory.mkdir();
// create a workspace
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
\end{verbatim}

See also \texttt{Transport3.java}.

7.10.3.10 How to read data from string and export to GDX (Transport3)

We read the data from the string data. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding \texttt{GAMSJob} a \texttt{GAMSDatabase} is created that is available via the \texttt{GAMSJob.OutDB} property. We can use the \texttt{GAMSDatabase.export} method to write the content of this database to a gdx file \texttt{tdata.gdx}.

\begin{verbatim}
// Create and run a job from a data file, then explicitly export to a GDX file
GAMSJob t3 = ws.addJobFromString(data);
t3.run();
t3.OutDB().export( ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gdx" );
\end{verbatim}

See also \texttt{Transport3.java}.
7.10 Tutorial

7.10.3.11 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in Transport2 - How to use include files (Transport2).

```java
// run a job using an instance of GAMSOptions that defines the data include file
t3 = ws.addJobFromString(model);
GAMSOptions opt = ws.addOptions();
opt.defines("gdxincname", "tdata");
t3.run(opt);
```

Note that there are some minor changes in model due to the usage of a gdx instead of an include file.

```java
if not set gdxincname

abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
```

See also Transport3.java.

7.10.3.12 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs t3a and t3b where the first one contains only the data and the second one contains only the model without data. After running t3a the corresponding OutDB can be read in directly just like a gdx file.

Note that the database needs to be passed to the GAMSJob.run method as additional argument.

```java
GAMSVariable x = t3a.OutDB().getVariable("x");
for (GAMSVariableRecord rec : x)
    System.out.println("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "): level=" + rec.getLevel() + " marginal=" + rec.getMarginal());
System.out.println();
// similar to the previous run but without exporting database into a file
GAMSJob t3a = ws.addJobFromString(data);
GAMSJob t3b = ws.addJobFromString(model);
opt = ws.addOptions();
t3a.run();
opt.defines("gdxincname", t3a.OutDB().getName());
t3b.run(opt, t3a.OutDB());
```

See also Transport3.java.

7.10.3.13 How to define data using Java data structures (Transport4)

We use the List<E> class and the Map<Key,Value> to define Java data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```java
// prepare input data
List<String> plants = Arrays.asList("Seattle", "San-Diego");
List<String> markets = Arrays.asList("New-York", "Chicago", "Topeka");
Map<String, Double> capacity = new HashMap<String, Double>();
{ capacity.put("Seattle", Double.valueOf(350.0));
capacity.put("San-Diego", Double.valueOf(600.0));
}
Map<String, Double> demand = new HashMap<String, Double>();
{ demand.put("New-York", Double.valueOf(325.0));
demand.put("Chicago", Double.valueOf(300.0));
demand.put("Topeka", Double.valueOf(275.0));
}
Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>();
{ distance.put(new Vector<String>(Arrays.asList("Seattle", "New-York")), Double.valueOf(2.5));
distance.put(new Vector<String>(Arrays.asList("Seattle", "Chicago")), Double.valueOf(1.7));
distance.put(new Vector<String>(Arrays.asList("Seattle", "Topeka")), Double.valueOf(1.8));
distance.put(new Vector<String>(Arrays.asList("San-Diego", "New-York")), Double.valueOf(2.5));
distance.put(new Vector<String>(Arrays.asList("San-Diego", "Chicago")), Double.valueOf(1.8));
distance.put(new Vector<String>(Arrays.asList("San-Diego", "Topeka")), Double.valueOf(1.4));
}
```

See also Transport4.java.
7.10.3.14 How to prepare a GAMSDatabase from Java data structures (Transport4)

At first we create an empty `GAMSDatabase db` using the `GAMSWorkspace.addDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.addSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Java data structure and add them by using the `GAMSSet.addRecord` method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that scalars can be treated as parameter with dimension 0.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```java
// add a database and add input data into the database
GAMSDatabase db = ws.addDatabase();
GAMSSet i = db.addSet("i", 1, "canning plants");
for(String p : plants)
i.addRecord(p);
GAMSSet j = db.addSet("j", 1, "markets");
for(String m : markets)
j.addRecord(m);
GAMSParameter a = db.addParameter("a", 1, "capacity of plant i in cases");
for (String p : plants)
a.addRecord(p).setValue( capacity.get(p) );
GAMSParameter b = db.addParameter("b", 1, "demand at market j in cases");
for(String m : markets)
b.addRecord(m).setValue( demand.get(m) );
GAMSParameter d = db.addParameter("d", 2, "distance in thousands of miles");
for(Vector<String> vd : distance.keySet())
d.addRecord(vd).setValue( distance.get(vd).doubleValue() );
GAMSParameter f = db.addParameter("f", 0, "freight in dollars per case per thousand miles");
f.addRecord().setValue( 90 );
// create and run a job from the model and read gdx include file from the database
GAMSJob t4 = ws.addJobFromString(model);
GAMSOptions opt = ws.addOptions();
opt.defines("gdxincname", db.getName());
t4.run(opt, db);
```

See also `Transport4.java`.

7.10.3.15 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following lines of code conduct several operations. While the first line simply creates a `GAMSCheckpoint`, the second one uses the `GAMSWorkspace.addJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. Afterwards the run method gets the `GAMSCheckpoint` as argument. That means the `GAMSCheckpoint cp` captures the state of the `GAMSJob`.

```java
GAMSCheckpoint cp = ws.addCheckpoint();
GAMSJob t5 = ws.addJobFromString(model);
t5.run(cp);
```

See also `Transport5.java`.

7.10.3.16 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `model` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```java
Scalar bmult demand multiplier /1/;
... demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...```
Scalar ms 'model status', ss 'solve status';

In **Transport5** we create a list with eight different values for this demand multiplier.

```java
    double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 }
```

For each entry of that list we create a **GAMSJob** `t5` using the **GAMSWorkspace.addJobFromString** method. Besides another string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This runs in a **GAMSJob** combined from the checkpoint plus the content provided by the string. We run the **GAMSJob** and echo some interesting data from the **OutDB** using the **GAMSParameter.getParameter** and **GAMSParameter.getVariable** methods, the **GAMSParameterRecord.getValue** property and the **GAMSVariableRecord.getLevel** method.

```java
    // create a new GAMSJob that is initialized from the GAMSCheckpoint
    for(double b : bmultlist)
    {
        t5 = ws.addJobFromString(
            "bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
            cp
        );
        t5.run();
        System.out.println("Scenario bmult=" + b + ":");
        System.out.println(\n            "Modelstatus: " + GAMSGlobals.ModelStat.lookup((int) t5.OutDB().getParameter("ms").findRecord().getValue())\n        );
        System.out.println(\n            "Solvestatus: " + GAMSGlobals.SolveStat.lookup((int)t5.OutDB().getParameter("ss").findRecord().getValue())\n        );
        System.out.println(" Obj: " + t5.OutDB().getVariable("z").findRecord().getLevel());
    }
```

Note
Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the **model status** and the **solve status** provide important information for a correct solution interpretation.

See also **Transport5.java**.

### 7.10.3.17 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs known from **Transport5** in parallel. We initialize the **GAMSCheckpoint** `cp` and introduce a demand multiplier as we did before:

```java
    GAMSJob t6 = ws.addJobFromString(model);
    t6.run(cp);
    double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 }
```

Furthermore, we introduce a new object `lockObject` that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of `bmultlist` and cause a thread to begin execution.

```java
    // run multiple parallel jobs using the created checkpoint
    Object lockObject = new Object();
    Scenario[] scenarios = new Scenario[bmultlist.length];
    for (int i=0; i<bmultlist.length; i++)
    {
        scenarios[i] = new Scenario(ws, cp, lockObject, bmultlist[i]);
        scenarios[i].start();
    }
```
In class Scenario a GAMSJob t6 is created and run just like in the preceding example of Transport5. The output section is also the same except for the fact that it is 'locked' by the object lockObject which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```java
public Scenario(GAMSWorkspace ws, GAMSCheckpoint cp, Object lockObj, double b) {
    workspace = ws;
    checkpoint = cp;
    lockObject = lockObj;
    bmult = b;
}
public void run() {
    GAMSJob t6 = workspace.addJobFromString(
        "bmult=" + bmult + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
        checkpoint
    );
    t6.run();
    // we need to make the output a critical section to avoid messed up report information
    synchronized (lockObject) {
        System.out.println("Scenario bmult=" + bmult + ":");
        System.out.println(" Modelstatus: " + GAMSGlobals.ModelStat.lookup(
            (int) t6.OutDB().getParameter("ms").findRecord().getValue()
        )
        );
        System.out.println(" Solvestatus: " + GAMSGlobals.SolveStat.lookup(
            (int)t6.OutDB().getParameter("ss").findRecord().getValue()
        )
        );
        System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
    }
}
```

While the output in Transport5 is strictly ordered subject to the order of the elements of bmultlist in Transport6 the output blocks might change their order but the blocks describing one scenario are still appearing together due to the lockObject.

If you want a further impression of the impact of the lockObject, just rerun Transport6 but comment out the lock as follows and compare the output.

```java
// synchronized (lockObject)
// {
    System.out.println("Scenario bmult=" + bmult + ":");
    System.out.println(" Modelstatus: " + GAMSGlobals.ModelStat.lookup(
        (int) t6.OutDB().getParameter("ms").findRecord().getValue()
    )
    );
    System.out.println(" Solvestatus: " + GAMSGlobals.SolveStat.lookup(
        (int)t6.OutDB().getParameter("ss").findRecord().getValue()
    )
    );
    System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
// }
```

See also Transport6.java.

7.10.3.18 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In Transport7 the usage of com::gams::api::GAMSModelInstance is demonstrated.
At first checkpoint \texttt{cp} is created as in the preceding examples. Then we create the \texttt{GAMSModelInstance mi} using the \texttt{GAMSCheckpoint.addModelInstance} method. Note that the \texttt{GAMSJob} again contains no solve statement and the demand multiplier is already included with default value 1.

```
GAMSCheckpoint cp = ws.addCheckpoint();
// initialize a checkpoint by running a job
GAMSJob t7 = ws.addJobFromString(model);
t7.run(cp);
GAMSModelInstance mi = cp.addModelInstance();
```

See also \texttt{Transport7.java}.

### 7.10.3.19 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A \texttt{GAMSModelInstance} uses a \texttt{SyncDB} to maintain the data. We define \texttt{bmult} as \texttt{GAMSParameter} using the \texttt{GAMSParameter} method and specify gurobi as solver. Afterwards the \texttt{GAMSModelInstance} is instantiated with 3 arguments, the solve statement, \texttt{GAMSOptions opt} and \texttt{GAMSModifier bmult}. The \texttt{GAMSModifier} means that \texttt{bmult} is modifiable while all other parameters, variables and equations of ModelInstance \texttt{mi} stay unchanged. We use the \texttt{GAMSParameter.addRecord} method and the \texttt{setValue} function to assign a value to \texttt{bmult}. That value can be varied afterwards using the \texttt{GAMSParameter.getFirstRecord} method to reproduce our well-known example with different demand multipliers.

```
GAMSParameter bmult = mi.SyncDB().addParameter("bmult", 0,"demand multiplier");
GAMSOptions opt = ws.addOptions();
opt.setAllModelTypes("gurobi");
// instantiate the ModelInstance and pass a model definition and Modifier to declare bmult mutable
mi.instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
bmult.addRecord().setValue(1.0);
double[] bmultlist = new double[]{0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3};
for (double b : bmultlist){
bmult.getFirstRecord().setValue(b);
System.out.println("Scenario bmult=": + b + ";
mi.solve();
```

See also \texttt{Transport7.java}.

### 7.10.3.20 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a \texttt{GAMSModelInstance} just like in the next to last example. We define \texttt{x} as \texttt{GAMSVariable} and its upper bound as \texttt{GAMSParameter xup}. At the following \texttt{instantiate} method \texttt{GAMSModifier} has 3 arguments. The first one says that \texttt{x} is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the \texttt{GAMSParameter} that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
mi = cp.addModelInstance();
GAMSVariable x = mi.SyncDB().addVariable("x", 2, GAMSGlobals.VarType.POSITIVE, "");
GAMSParameter xup = mi.SyncDB().addParameter("xup", 2, "upper bound on x");
// instantiate the ModelInstance and pass a model definition and Modifier to declare upper bound of X mutable
mi.instantiate("transport use lp min z", new GAMSModifier(x, GAMSGlobals.UpdateAction.UPPER, xup));
for (GAMSSetRecord i : t7.OutDB().getSet("i")){
    for (GAMSSetRecord j : t7.OutDB().getSet("j"){
```
3214 Application Programming Interfaces

```java
xup.clear();
String[] keys = { i.getKeys()[0], j.getKeys()[0] };
xup.addRecord(keys).setValue(0);
mi.solve();
System.out.println("Scenario link blocked: " + i.getKeys()[0] + " - " + j.getKeys()[0]);
System.out.println("Modelstatus: " + mi.getModelStatus());
System.out.println("Solvestatus: " + mi.getSolveStatus());
System.out.println("Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
}
```

See also Transport7.java.

7.10.3.21 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint cp from a GAMSJob. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out principle. The object IOLockObject is used later to avoid messed up output. Then we call Scenarios multiple times in parallel. The number of parallel calls is specified by numberOfWorkers.

```java
GAMSJob t8 = ws.addJobFromString(model);
t8.run(cp);
Queue<Double> bmultQueue = new LinkedList<Double>(
    Arrays.asList(Double.valueOf(0.6), Double.valueOf(0.7),
                 Double.valueOf(0.8), Double.valueOf(0.9),
                 Double.valueOf(1.0), Double.valueOf(1.1),
                 Double.valueOf(1.2), Double.valueOf(1.3))
);
// solve multiple model instances in parallel
Object IOLockObject = new Object();
int numberOfWorkers = 2;
Scenarios[] scenarios = new Scenarios[numberOfWorkers];
for (int i=0; i<numberOfWorkers; i++)
{
    scenarios[i] = new Scenarios( ws, cp, bmultQueue, IOLockObject, i );
    scenarios[i].start();
}
```

In class Scenarios we create and instantiate a GAMSModelInstance as in the preceding examples and make parameter bmult modifiable. Note that we chose cplex as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```java
static class Scenarios extends Thread
{
    GAMSWorkspace workspace;
    GAMSCheckpoint checkpoint;
    Object IOLockObject;
    Queue<Double> bmultQueue;
    int workerNumber;
    public Scenarios(GAMSWorkspace ws, GAMSCheckpoint cp, Queue<Double> que, Object IOLockObj, int i)
    {
        workspace = ws;
        checkpoint = cp;
        IOLockObject = IOLockObj;
        bmultQueue = que;
        workerNumber = i;
    }
    public void run()
    {
        GAMSModelInstance mi = checkpoint.addModelInstance();
        {
            GAMSParameter bmult = mi.SyncDB().addParameter("bmult", 0, "demand multiplier");
            GAMSOptions opt = workspace.addOptions();
            opt.setAllModelTypes("cplex");
            // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
            mi.instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
            bmult.addRecord().setValue(1.0);
            while (true)
                double b = 0.0;
                // dynamically get a bmult value from the queue instead of passing it to the different threads at creation time
                synchronized (bmultQueue)
                    if (bmultQueue.isEmpty())
                        break;
        }
```
7.10 Tutorial

7.10.3.22 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a GAMSDatabase (on Windows platform only). There are a few prerequisites required to run Transport9 successfully.

- Install a Microsoft access driver on your machine (if none is available) as the driver must be loaded at runtime to connect to a data source file.

- To access the data source file go to Control Panel\System and Security\Administrative Tools\Data Sources (ODBC) and set up a data source name (DSN) as "transportdsn", leave out user ID and password, and select [PathToGAMS]/apfiles/Data/tansport.accdb as the data source file.

- Note that an architecture mismatch between the Driver and Application might cause problems.

We call a function readDataFromAccess that finally returns a GAMSDatabase as shown below.

```java
GAMSDatabase db = readDataFromAccess(ws);
...
```

The function readDataFromAccess begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database by specifying the aforementioned data source name (DSN). To finally read in GAMS sets and parameters we call the functions readSet and readParameter.

```java
static GAMSDatabase readDataFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.addDatabase();
    try {
        // loading the jdbc odbc driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        // creating connection to database
        Connection c = DriverManager.getConnection("jdbc:odbc:transportdsn", "", "");
        // read GAMS sets
        readSet(c, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
        readSet(c, db, "SELECT Market FROM Market", "j", 1, "markets");
        // read GAMS parameters
        readParameter(c, db, "SELECT Plant, Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
        readParameter(c, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
        readParameter(c, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles");
        c.close();
    } catch (ClassNotFoundException e) {
        System.err.println("Error: Failed to find a driver for the database.");
e.printStackTrace();
        System.exit(-1);
    } catch (SQLException e) {
        System.err.println("Error: Failed to retrieve data from the database.");
e.printStackTrace();
        System.exit(-1);
    }    return db;
}
```

See also Transport8.java.
The function `readSet` adds a set to the `GAMSDatabase` that is filled with the data from the MS Access file afterwards. The function `readParameter` works quite similar.

```java
static void readSet(Connection c, GAMSDatabase db, String queryString, String setName, int setDimension, String setExplanatoryText) throws SQLException {
    Statement st = c.createStatement();
    ResultSet rs = st.executeQuery(queryString);
    ResultSetMetaData rsmd = rs.getMetaData();
    if (rsmd.getColumnCount() != setDimension) {
        System.err.println("Error: Number of fields in select statement does not match setDimension.");
        c.close();
        System.exit(-1);
    }
    GAMSSet set = db.addSet(setName, setDimension, setExplanatoryText);
    String[] keys = new String[setDimension];
    while (rs.next()) {
        for (int idx=0; idx < setDimension; idx++)
            keys[idx] = rs.getString(idx+1);
        set.addRecord(keys);
    }
    st.close();
}
```

Once we read in all the data we can create a `GAMSJob` from the `GAMSDatabase` and run it as usual.

See also `Transport9.java`.

### 7.10.3.23 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `\[PathToGAMS\]/apifiles/Data/transport.xlsx`. At first we have to add an additional jar file to the `[CLASSPATH]` since we use the Java Excel API which can be downloaded from [jexcelapi](https://jexcelapi.org). We unzipped the Java Excel API folder in `C:/tools`. To compile and run on a Windows platform from the command line go to `C:/GAMS/win64/24.1/apifiles/Java/transport` and use:

```bash
javac -cp C:/GAMS/win64/GAMS_VERSION/apifiles/Java/api/GAMSJavaAPI.jar;C:/tools/jexcelapi/jxl.jar;. -d . Transport10.java
```

and

```bash
java -cp C:/GAMS/win64/GAMS_VERSION/apifiles/Java/api/GAMSJavaAPI.jar;C:/tools/jexcelapi/jxl.jar;. com.gams.examples.transport.Transport10
```

If you are using an IDE like Eclipse add `C:\tools\jexcelapi\jxl.jar` to the class path as explained in the [Compiling a Program from Java IDE and Running a Program from Java IDE section](#).

The model is given as string without data like in many examples before and the Excel file `transport.xlsx` is located at `[gamsdir]/apifiles/Data`. At first we define the input string and create the corresponding input file.

```java
String input = gamsdir + "apifiles" +
    GAMSGlobals.FILE_SEPARATOR + "Data" + GAMSGlobals.FILE_SEPARATOR + "transport.xlsx";
File inputFile = new File(input);
```

The following lines address the different worksheets and read in the contained data.

```java
int iCount = 0;
int jCount = 0;
String[][] capacityData = null;
String[][] demandData = null;
String[][] distanceData = null;
Workbook w;
```
try {
    w = Workbook.getWorkbook(inputFile);
    Sheet capacity = w.getSheet("capacity");
    capacityData = new String[capacity.getRows()][capacity.getColumns()];
    iCount = capacity.getColumns();
    for (int j = 0; j < capacity.getColumns(); j++)
        for (int i = 0; i < capacity.getRows(); i++)
            capacityData[i][j] = capacity.getCell(j, i).getContents();
    Sheet demand = w.getSheet("demand");
    demandData = new String[demand.getRows()][demand.getColumns()];
    jCount = demand.getColumns();
    for (int j = 0; j < demand.getColumns(); j++)
        for (int i = 0; i < demand.getRows(); i++)
            demandData[i][j] = demand.getCell(j, i).getContents();
    Sheet distance = w.getSheet("distance");
    distanceData = new String[distance.getRows()][distance.getColumns()];
    for (int j = 0; j < distance.getColumns(); j++)
        for (int i = 0; i < distance.getRows(); i++)
            distanceData[i][j] = distance.getCell(j, i).getContents();
    w.close();
} catch (IOException e) {
    e.printStackTrace();
} catch (BiffException e) {
    e.printStackTrace();
}

Now we can create the GAMSWorkspace as usual and afterwards create a GAMSDatabase and fill it with the workbook data as follows:

```java
GAMSDatabase db = ws.addDatabase();
GAMSSet i = db.addSet("i", 1, "Plants");
GAMSSet j = db.addSet("j", 1, "Markets");
GAMSParameter capacityParam = db.addParameter("a", 1, "Capacity");
GAMSParameter demandParam = db.addParameter("b", 1, "Demand");
GAMSParameter distanceParam = db.addParameter("d", 2, "Distance");
for (int ic = 0; ic < iCount; ic++)
    i.addRecord( capacityData[0][ic] );
    capacityParam.addRecord( capacityData[0][ic] ).setValue( Double.valueOf(capacityData[1][ic]).doubleValue() );
for (int jc = 0; jc < jCount; jc++)
    j.addRecord( demandData[0][jc] );
    demandParam.addRecord( demandData[0][jc] ).setValue( Double.valueOf(demandData[1][jc]).doubleValue() );
    String[] data = null;
    for (int ic = 0; ic < iCount; ic++)
        data = new String[] {
            distanceData[ic+1][0], distanceData[0][jc+1]};
        distanceParam.addRecord( data ).setValue( Double.valueOf(distanceData[ic+1][jc+1]) );
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we can run our GAMSJob as usual.

```java
GAMSOptions opt = ws.addOptions();
GAMSJob t10 = ws.addJobFromString(model);
opt.defines( "gdxincname", db.getName() );
opt.setAllModelTypes( "xpress" );
t10.run( opt, db );
for (GAMSVariableRecord rec : t10.OutDB().getVariable("x") )
    System.out.println("x(" + rec.getKeys()[0] + "," + rec.getKeys()[1] + "):: level=" + rec.getLevel() + " marginal=" + rec.getMarginal() );
```

See also Transport10.java.

7.10.3.24 How to create and use a save/restart file (Transport11)

In Transport11 we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a GAMSCheckpoint. From the main function we call the function CreateSaveRestart giving it the current working directory and the desired file name as arguments.

```java
CreateSaveRestart( workingDirectory, "tbase" );
```
In function `CreateSaveRestart` create a workspace with the given working directory. Then we create a `GAMSJob` from a string. Note that the string given via `baseModel` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a `GAMSOptions` to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the `GAMSJob` and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the `addCheckpoint` method accepts identifiers as well as file names as argument.

```java
static void CreateSaveRestart(File workingDirectory, String cpFileName)
{
    GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
    wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
    GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
    GAMSJob j1 = ws.addJobFromString(baseModel);
    GAMSOptions opt = ws.addOptions();
    opt.setAction( GAMSOptions.EAction.CompileOnly );
    GAMSCheckpoint cp = ws.addCheckpoint(cpFileName);
    j1.run(opt, cp);
    opt.dispose();
}
```

So what you should keep in mind before we return to further explanations of the main function is, that the file `tbase` is now in the current working directory and contains a checkpoint. Now in the main function we define some data using Java data structures as we already did in `Transport4` before we create the `GAMSWorkspace` and a `GAMSDatabase`.

```java
...
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
GAMSDatabase db = ws.addDatabase();
...
```

Afterwards we set up the `GAMSDatabase` like we already did in `Transport4`. Once this is done we run a `GAMSJob` using this data plus the checkpoint stored in file `tbase`.

```java
...
GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");
GAMSOptions opt = ws.addOptions();
GAMSJob t11 = ws.addJobFromString(model, cpBase);
opt.defines("gdxincname", db.getName());
opt.setAllModelTypes("xpress");
t11.run(opt, db);
...
```

Note that the string from which we create job `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the `GAMSDatabase` correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

See also `Transport11.java`.

### 7.11 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Python API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known transportation problem. These examples introduce several API features step by step.

- **Getting started** A quick introduction about how to run a program
- **Important Classes of the API** Overview of some fundamental classes of the GAMS Python API
- **How to use the API** An extensive set of examples describing how to use API components
7.11 Tutorial 3219

7.11.1 Getting started

The object oriented GAMS Python API is built on top of the different expert-level APIs and provides convenient access to GAMS from within Python. Examples using the API are located in `apifiles/Python`. The expert-level APIs can be found in `apifiles/Python/api_37` for Python 3.7, in `apifiles/Python/api_38` for Python 3.8, in `apifiles/Python/api_39` for Python 3.9, and in `apifiles/Python/api_310` for Python 3.10. The object oriented API is located in `apifiles/Python/gams`. Making your Python installation aware of the modules provided by GAMS can be done in two fundamentally different ways. Which method to choose depends on your preference and your use pattern of GAMS and Python. Note also that the bitness of your Python version has to match the bitness of the GAMS system.

7.11.1.1 Let Python find the GAMS Python files

There are many ways for a Python installation to find locations with additional packages. Here are two:

a) Extend/set the PYTHONPATH environment variable

The environment variable PYTHONPATH (https://docs.python.org/3/using/cmdline.html#environment-variables) can be used to let Python find the GAMS API.

macOS/Linux:

```bash
export PYTHONPATH=/path/to/gams/apifiles/Python/api_38
export PYTHONPATH=/path/to/gams/apifiles/Python/gams:$PYTHONPATH
```

Windows:

```bash
set PYTHONPATH=C:\path\to\gams\apifiles\Python\api_38
set PYTHONPATH=C:\path\to\gams\apifiles\Python\gams;%PYTHONPATH%
```

b) Use the site package

The site package (https://docs.python.org/3/library/site.html) can be used to include the GAMS directories. For example, create a file `sitecustomize.py` in your site-packages directory with the following content:

```python
import sys
sys.path.append(r'C:\path\to\gams\apifiles\Python\api_38')
sys.path.append(r'C:\path\to\gams\apifiles\Python\gams')
```

Change the path separator to / for macOS and Linux. You find out where your site-packages directory is located by running `python -m site` and inspecting the output. Method b) is preferred if creating permanent environment variables is difficult (e.g. on macOS).
Copy the GAMS Python files to the Python installation

Open a shell/command prompt and make sure your Python interpreter is found. Change directories to your GAMS system directory with the Python 3.X API files and run `python setup.py install`, e.g. on Windows:

```
cd C:\path\to\gams\apifiles\Python\api38
C:\path\to\my\python\python.exe setup.py install
```

**Attention**

With `setuptools>=60.0.0` a breaking change has been introduced that causes the GAMS Python files to be installed incorrectly. In this case it is required to set the environment variable `SETUPTOOLS_USE_DISTUTILS` to `stdlib` before running `setup.py`.

**macOS/Linux:**

```
export SETUPTOOLS_USE_DISTUTILS=stdlib
```

**Windows:**

```
set SETUPTOOLS_USE_DISTUTILS=stdlib
```

**Note**

Per default, the `setup.py` script tries to create a `build` directory before installing its content into the `site-packages` directory of the Python interpreter. In case of missing write permissions in the GAMS system directory (e.g. on macOS) it is possible to specify an alternative location using the following command:

```
C:\path\to\my\python\python.exe setup.py build -b C:\path\to\build install
```

**Note**

For the use of the Embedded Code Facility for Python, certain dependencies might be required (e.g. `ply`). The directory `/path/to/gams/apifiles/Python/thirdparty` contains all the third party modules used by GMSPython. The content of this directory indicates which modules and packages should be installed in addition.

**Running the transport1.py example:**

In order to see that the Python installation is set up correctly to be used with GAMS, a simple test is to run the `transport1.py` example. Note that you have to change the path separator to `/` for macOS and Linux:

```
C:\path\to\my\python\python.exe C:\path\to\gams\apifiles\Python\transport1.py
```

Working with Python and multiple GAMS Installations

GAMS uses SWIG ([http://www.swig.org/](http://www.swig.org/)) to connect Python to GAMS functionality implemented in dynamic link libraries or shared objects (e.g. `gdxdclib64.dll` to access GDX functionality). GAMS provides all files necessary to access these libraries from Python in the `apifiles/Python/api_3X` directories. Due to changes in the library API, SWIG version, Python version, etc, these files, that you might have copied to your Python installation in case you choose the copy method from above, are highly GAMS distribution dependent. If you work with a single GAMS installation this is very easy but if you need to work with different GAMS installations connecting your own Python installation can be a challenge. If you are in this situation we highly recommend to work with a Python installation that supports environment management, like Conda. Create for each major GAMS version a new environment and choose the copy method or the site customization (see entry b)) (e.g. via `sitecustomize.py`) to connect GAMS with this environment. Here is a typical session to create two environments, one for GAMS 31 and one for GAMS 32 using Conda:

```
conda create --name gams31 python=3.8
conda activate gams31
cd /path/to/gams31/apifiles/Python/api38
python setup.py install
conda create --name gams32 --clone gams31
conda activate gams32
cd /path/to/gams32/apifiles/Python/api38
python setup.py install
```
7.11 Tutorial

7.11.1.4 Specifying a GAMS System Directory

There are several ways to specify which system directory should be used. On all platforms, the system directory can be specified in the GamsWorkspace constructor. If no system directory is specified by the user, the API tries to find one automatically:

- Windows: Try to find a system directory in the Windows registry.
- Linux: Try to find a system directory in the PATH first. If none was found, search LD_LIBRARY_PATH.
- OS X: Try to find a system directory in the PATH first. If none was found, search DYLD_LIBRARY_PATH.

The environment variable PATH can be set as follows on Linux and macOS:

```
export PATH=<Path/To/GAMS>:
```

Note

On Linux and macOS it is recommended to specify the PATH only instead of (DY)LD_LIBRARY_PATH since this might cause problems loading the correct version of certain modules (e.g. gdx).

7.11.1.5 Troubleshooting

AttributeError: module 'gams' has no attribute 'SuffixDLVars':

Since GAMS 31, the Object-oriented Python API is distributed as source code (*.py files) instead of binaries (*.pyd files on Windows, *.so files on Linux and macOS). Installing the API with python setup.py install copies the API files into the site-packages directory of the Python interpreter, but won’t overwrite or delete old *.so/*.pyd files since those are not part of the distribution anymore. This might lead to wrongly loaded files that are not compatible with each other anymore (e.g. options.py).

In order to resolve the problem it is required to delete the gams directory located in the site-packages directory of the used Python installation before installing a new version of the API. Executing python -m site will reveal the location of the site-packages directory.

7.11.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples in the How to use the API section.

- `gams::workspace::GamsWorkspace` Class
- `gams::execution::GamsJob` Class
- `gams::database::GamsDatabase` Class
- `gams::options::GamsOptions` Class
- `gams::execution::GamsModelInstance` Class
7.11.3 How to use the API

The GAMS distribution provides several examples that illustrate the usage of the Python API. \texttt{[GAMSDIR]/apifiles/Python} contains multiple examples dealing with the well-known \texttt{transportation problem}. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to import packages/modules from the GAMS Python API (transport1.py)
- How to choose the GAMS system (transport1.py)
- How to export data to GDX (transport_gdx.py)
- How to import data from GDX (transport_gdx.py)
- How to run a GamsJob from file (transport1.py)
- How to retrieve a solution from an output database (transport1.py)
- How to specify the solver using GamsOptions (transport1.py)
- How to run a job with a solver option file and capture its log output (transport1.py)
- How to use include files (transport2.py)
- How to read data from string and export to GDX (transport3.py)
- How to run a job using data from GDX (transport3.py)
- How to run a job using implicit database communication (transport3.py)
- How to define data using Python data structures (transport4.py)
- How to prepare a GamsDatabase from Python data structures (transport4.py)
- How to initialize a GamsCheckpoint by running a GamsJob (transport5.py)
- How to initialize a GamsJob from a GamsCheckpoint (transport5.py)
- How to run multiple GamsJobs in parallel using a GamsCheckpoint (transport6.py)
- How to create a GamsModelInstance from a GamsCheckpoint (transport7.py)
- How to modify a parameter of a GamsModelInstance using GamsModifier (transport7.py)
- How to modify a variable of a GamsModelInstance using GamsModifier (transport7.py)
- How to use a queue to solve multiple GamsModelInstances in parallel (transport8.py)
- How to fill a GamsDatabase by reading from MS Access (transport9.py)
- How to fill a GamsDatabase by reading from MS Excel (transport10.py)
- How to create and use a save/restart file (transport11.py)

7.11.3.1 How to import packages/modules from the GAMS Python API (transport1.py)

If you followed the instructions from the \texttt{Getting started} section and installed the GAMS Python API there should be a directory \texttt{[PYTHONDIR]\Lib\site-packages\gams} that contains the required packages and modules. We simply import all of them using

\begin{verbatim}
from gams import *
\end{verbatim}

Conventional Python packages/modules can by imported like that:

\begin{verbatim}
import os
import sys
\end{verbatim}
7.11 Tutorial

7.11.3.2 How to choose the GAMS system (transport1.py)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. If we type `python transport1.py C:/GAMS/win64/40.1` we use the 64-bit version of GAMS 40.1 to run `transport1.py` even if our default GAMS system might be a different one. This is managed by the following code:

```python
if len(sys.argv) > 1:
    ws = GamsWorkspace(system_directory = sys.argv[1])
else:
    ws = GamsWorkspace()
```

Remember that the bitness of the GAMS system has to match the bitness of your Python version.

7.11.3.3 How to export data to GDX (transport_gdx.py)

Although the Object-oriented Python API offers much more than exchanging data between Python and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GamsDatabase. We assume that the data to be exported is available in Python data structures.

```python
plants = ["Seattle", "San-Diego"]
markets = ["New-York", "Chicago", "Topeka"]
capacity = {
    "Seattle": 350.0,  "San-Diego": 600.0
}
demand = {
    "New-York": 325.0,  "Chicago": 300.0,  "Topeka": 275.0
}
distance = {
    ("Seattle", "New-York") : 2.5,
    ("Seattle", "Chicago") : 1.7,
    ("Seattle", "Topeka") : 1.8,
    ("San-Diego", "New-York") : 2.5,
    ("San-Diego", "Chicago") : 1.8,
    ("San-Diego", "Topeka") : 1.4
}
```

Different GAMS symbols are represented using different Python data structures. The data for the GAMS sets is represented using Python lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by Python dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses Python tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with standard Python data structures.

A new `GamsDatabase` instance can be created using `GamsWorkspace.add_database`.

```python
# create new GamsDatabase instance
db = ws.add_database()
```

We start adding GAMS sets using the method `GamsDatabase.add_set` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created `GamsSet` instance `i` using `GamsSet.add_record`.

```python
# add 1-dimensional set 'i' with explanatory text 'canning plants' to the GamsDatabase
i = db.add_set("i", 1, "canning plants")
for p in plants:
    i.add_record(p)
```

GamsParameter instances can be added by using the method `GamsDatabase.add_parameter`. It has the same signature as `GamsDatabase.add_set`. Anyhow, in this example we use `GamsDatabase.add_parameter_dc` instead which takes a list of `GamsSet` instances instead of the dimension for creating a parameter with domain information.

```python
# add parameter 'a' with domain 'i'
a = db.add_parameter_dc("a", [i], "capacity of plant i in cases")
for p in plants:
    a.add_record(p).value = capacity[p]
```

As soon as all data is prepared in the `GamsDatabase`, the method `GamsDatabase.export` can be used to create a GDX file.

```python
# export the GamsDatabase to a GDX file with name 'data.gdx' located in the 'working directory' of the GamsWorkspace
db.export("data.gdx")
```
7.11.3.4 How to import data from GDX (transport_gdx.py)

Data can be imported from a GDX file using GamsWorkspace.add_database_from_gdx. The method takes a path to a GDX file and creates a GamsDatabase instance.

```
# add a new GamsDatabase and initialize it from the GDX file just created
db2 = ws.add_database_from_gdx('data.gdx')
```

Reading the data from the GamsSet i into a list can be done as follows:

```
# read data from symbols into Python data structures
i = [ rec.keys[0] for rec in db2['i'] ]
```

A Python list is created using list comprehensions. i is retrieved by querying the GamsDatabase db2. The returned GamsSet object can be iterated using a for-loop to access the records of the set. Each record is of type GamsSetRecord and can be asked for its keys.

You can do the same for GamsParameters. Instead of creating a Python list, we want to have the data in the form of a Python dictionary. GamsParameterRecords can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter a into a Python dictionary using dict comprehensions.

```
a = { rec.keys[0]:rec.value for rec in db2['a'] }
```

For multi dimensional symbols, we choose the Python dictionary keys to be tuples instead of string. We access the keys as usual, but do not address a specific key. Instead, we take the whole list of keys and turn it into a tuple.

```
d = dict( (tuple(rec.keys), rec.value) for rec in db2['d'] )
```

 Scalars can be read into a Python identifier by accessing the value of the first and only record.

```
f = db['f'].first_record().value
```

7.11.3.5 How to run a GamsJob from file (transport1.py)

At first we create our workspace using `ws = GamsWorkspace()`. Afterwards we load the transport model from the GAMS model library which puts the corresponding gms file in our working directory. Apparently you can create a GamsJob with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GamsJob t1 can be defined using the add_job_from_file method and afterwards we run the job.

```
ws.gamslib('trnsport')
t1 = ws.add_job_from_file('trnsport.gms')
t1.run()
```

7.11.3.6 How to retrieve a solution from an output database (transport1.py)

The following lines create the solution output and illustrate the usage of the GamsJob.out_db property to get access to the GamsDatabase created by the run method. To retrieve the content of variable x we use squared brackets that internally call the get_symbol method.

```
for rec in t1.out_db['x']:
    print "%s" % rec.key() + "," + rec.value(): level=" + str(rec.level) + " marginal=" + str(rec.marginal)
```

Note that instead of using the squared brackets we could also use

```
for rec in t1.out_db.get_symbol('x'):
    ...
```
7.11 Tutorial

7.11.3.7 How to specify the solver using GamsOptions (transport1.py)

The solver can be specified via the GamsOptions class and the GamsWorkspace.add_options method. The GamsOptions.all_model_types property sets xpress as default solver for all model types that can be handled by the solver. Then we run our GamsJob t1 with the new GamsOption.

```python
... 
opt = ws.add_options()
opt.all_model_types = "xpress"
t1.run(opt)
...
```

7.11.3.8 How to run a job with a solver option file and capture its log output (transport1.py)

At first we create the file xpress.opt with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterwards we use a GamsOption just like in the preceding example and set GamsOption.optfile property to 1 to tell the solver to look for a solver option file. In addition, we specify the argument output in order to write the log of the GamsJob into the file `transport1_xpress.log`. Providing `newline=""` to the open function prevents additional newline characters in the log file.

```python
file = open(os.path.join(ws.working_directory, "xpress.opt"), "w")
file.write("algorithm=barrier")
file.close()
... 
opt.optfile = 1
with open("transport1_xpress.log", "w", newline="") as log:
t1.run(opt, output=log)
...
```

Instead of writing the log output to a file, any object that provides the functions write and flush can be used. In order to write the log directly to stdout, we can use the following code:

```python
... 
t1.run(opt, output=sys.stdout)
...
```

7.11.3.9 How to use include files (transport2.py)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings accessed via functions get_data_text and get_model_text are using GAMS syntax. At first we write an include file tdata.gms that contains the data but not the model text and save it in our current working directory.

```python
file = open(os.path.join(ws.working_directory, "tdata.gms"), "w")
file.write(get_data_text())
file.close()
...
```

Afterwards we create a GamsJob using the GamsWorkspace.add_job_from_string method. GamsOptions.defines is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where incname is used as name for the include file in get_model_text as shown below.

```python
... 
t2 = ws.add_job_from_string(get_model_text())
opt = ws.add_options()
opt.defines["incname"] = "tdata"
t2.run(opt)
...
```

The string get_model_text contains the following lines to read in the data.

```gams
$if not set incname $abort 'no include file name for data file provided'
$include [incname] 
```
7.11.3.10 How to read data from string and export to GDX (transport3.py)

We read the data from the string accessed via function `get_data_text`. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding GamsJob a GamsDatabase is created that is available via the GamsJob.out.db property. We can use the GamsDatabase.export method to write the content of this database to a gdx file `tdata.gdx` in the current working directory.

```python
... t3 = ws.add_job_from_string(get_data_text())
    t3.run()
    t3.out_db.export(os.path.join(ws.working_directory, "tdata.gdx"))
... 
```

7.11.3.11 How to run a job using data from GDX (transport3.py)

This works quite similar to the usage of an include file explained in How to use include files (transport2.py).

```python
... t3 = ws.add_job_from_string(get_model_text())
opt = ws.add_options()
    opt.defines["gdxincname"] = "tdata"
    opt.all_model_types = "xpress"
    t3.run(opt)
... 
```

Note that there are some minor changes in `get_model_text` compared to preceding examples due to the usage of a gdx instead of an include file.

```plaintext
$if not set gdxincname $abort 'no include file name for data file provided'
.gdxin %gdxincname%
.gdxin
... 
```

7.11.3.12 How to run a job using implicit database communication (transport3.py)

This example does basically the same as the two preceding examples together. We create two GamsJobs `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding `out.db` can be read in directly just like a gdx file. Note that the database needs to be passed to the GamsJob.run method as additional argument.

```python
... t3a = ws.add_job_from_string(get_data_text())
    t3a.run()
    opt.defines["gdxincname"] = t3a.out_db.name
    t3b.run(opt, databases=t3a.out_db)
... 
```

7.11.3.13 How to define data using Python data structures (transport4.py)

We use squared brackets (Python Lists) to define the sets and curly brackets (Python Dictionaries) for the parameter definition.

```python
... plants = [ "Seattle", "San-Diego" ]
markets = [ "New-York", "Chicago", "Topeka" ]
capacity = { "Seattle": 350.0, "San-Diego": 600.0 }
demand = { "New-York": 325.0, "Chicago": 300.0, "Topeka": 275.0 }
distance = { ("Seattle", "New-York") : 2.5,
            ("Seattle", "Chicago") : 1.7,
            ("Seattle", "Topeka") : 1.8,
            ("San-Diego", "New-York") : 2.5,
            ("San-Diego", "Chicago") : 1.8,
            ("San-Diego", "Topeka") : 1.4 }
... 
```
7.11 Tutorial

7.11.3.14 How to prepare a GamsDatabase from Python data structures (transport4.py)

At first we create an empty GamsDatabase db using the GamsWorkspace.add_database method. Afterwards we prepare the database. To add a set to the database we use the GamsSet class and the GamsDatabase.add_set method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Python data structure and add them by using the GamsSet.add_record method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The GamsJob can be run like explained in the preceding example How to run a job using implicit database communication...

```
... 
  db = ws.add_database() 
  i = db.add_set("i", 1, "canning plants") 
  for p in plants: 
    i.add_record(p) 
  j = GamsSet(db, "j", 1, "markets") 
  for m in markets: 
    j.add_record(m) 
  a = GamsParameter(db, "a", 1, "capacity of plant i in cases") 
  for p in plants: 
    a.add_record(p).value = capacity[p] 
  b = GamsParameter(db, "b", 1, "demand at market j in cases") 
  for m in markets: 
    b.add_record(m).value = demand[m] 
  d = GamsParameter(db, "d", 2, "distance in thousands of miles") 
  for k, v in distance.items(): 
    d.add_record(k).value = v 
  f = GamsParameter(db, "f", 0, "freight in dollars per case per thousand miles") 
  f.add_record().value = 90 
  t4 = GamsJob(ws, source=get_model_text()) 
  opt = GamsOptions(ws) 
  opt.defines["gdxincname"] = db.name 
  opt.all_model_types = "xpress" 
  t4.run(opt, databases = db) 
... 
```

7.11.3.15 How to initialize a GamsCheckpoint by running a GamsJob (transport5.py)

The following lines of code conduct several operations. While the first line simply creates a GamsCheckpoint, the second one uses the GamsWorkspace.add_job_from_string method to create a GamsJob containing the model text and data but no solve statement. Afterwards the run method gets the GamsCheckpoint as argument. That means the GamsCheckpoint cp captures the state of the GamsJob.

```
... 
  cp = ws.add_checkpoint() 
  t5 = ws.add_job_from_string(get_model_text()) 
  t5.run(checkpoint=cp) 
... 
```

7.11.3.16 How to initialize a GamsJob from a GamsCheckpoint (transport5.py)

Note that the string returned from function get_model_text contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
Scalar bmult demand multiplier /1/; 
... 
  demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ; 
... 
Scalar ns 'model status', ss 'solve status'; 
... 
```

In transport5.py we create a list with eight different values for this demand multiplier.

```
... 
  bmultlist = [ 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 ] 
... 
```

For each entry of that list we create a GamsJob t5 using the GamsWorkspace.add_job_from_string method. Besides another string which resets the demand multiplier bmult, specifies the solve statement and assigns
values to the scalars \( ms \) and \( ss \) we pass the checkpoint \( cp \) as additional argument. This results in a GamsJob combined from the checkpoint plus the content provided by the string.

We run the GamsJob and print some interesting data from the \texttt{out_db}.

```python
for b in bmultlist:
    t5 = ws.add_job_from_string("bmult='{}'; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;", cp)
    t5.run()
    print "Scenario bmult='{}':".
    print " Modelstatus: " + str(t5.out_db["ms"].find_record().value)
    print " Solvestatus: " + str(t5.out_db["ss"].find_record().value)
    print " Obj: " + str(t5.out_db["z"].find_record().level)
```

NOTE: Some of the demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

### 7.11.3.17 How to run multiple GamsJobs in parallel using a GamsCheckpoint (transport6.py)

This example illustrates how to run the jobs known from transport5.py in parallel. We initialize the GamsCheckpoint \( cp \) and introduce a demand multiplier as we did before:

```python
cp = ws.add_checkpoint()
t6 = ws.add_job_from_string(get_model_text())
t6.run(checkpoint=cp)
bmultlist = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3]
```

Furthermore, we introduce a lock object \texttt{io_lock} that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of \texttt{bmultlist} and cause a thread to begin execution.

```python
io_lock = threading.Lock()
threads = {}
for b in bmultlist:
    threads[b] = threading.Thread(target=run_scenario, args=(ws, cp, io_lock, b))
    threads[b].start()
for b in bmultlist:
    threads[b].join()
```

In function \texttt{run_scenario} a GamsJob \( t6 \) is created and run just like in the preceding example of transport5.py. The output section is also the same except for the fact that it is 'locked' by the object \texttt{io_lock} which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```python
def run_scenario(workspace, checkpoint, io_lock, b):
    t6 = workspace.add_job_from_string("bmult='{}'; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;", checkpoint)
    t6.run()
    io_lock.acquire()
    print "Scenario bmult='{}':".
    print " Modelstatus: " + str(t6.out_db["ms"].value)
    print " Solvestatus: " + str(t6.out_db["ss"].value)
    print " Obj: " + str(t6.out_db["z"].level)
    io_lock.release()
```

While the output in transport5.py is strictly ordered subject to the order of the elements of \texttt{bmultlist} in transport6.py the output blocks might change their order but the blocks describing one scenario are still appearing together due to the \texttt{io_lock} object.
7.11 Tutorial

7.11.3.18 How to create a GamsModelInstance from a GamsCheckpoint (transport7.py)

In transport7.py the usage of GamsModelInstance is demonstrated.

At first checkpoint cp is created as in the preceding examples. Note that the GamsJob t7 again contains no solve statement and the demand multiplier is already included with default value 1. We create the GamsModelInstance mi using the GamsCheckpoint.add_modelinstance method.

```
... cp = ws.add_checkpoint()
t7 = ws.add_job_from_string(get_model_text())
t7.run(checkpoint=cp)
mi = cp.add_modelinstance()
...```

7.11.3.19 How to modify a parameter of a GamsModelInstance using GamsModifier (transport7.py)

A GamsModelInstance uses a sync_db to maintain the data. We define bmult as GamsParameter using the GamsDatabase.add_parameter method and specify gurobi as solver. Afterwards the GamsModelInstance is instantiated with 3 arguments, the solve statement, GamsModifier bmult and GamsOptions opt. The GamsModifier means that bmult is modifiable while all other parameters, variables and equations of GamsModelInstance mi stay unchanged. We use the GamsParameter.add_record method to assign a value to bmult. That value can be varied afterwards using the GamsParameter.first_record method to reproduce our well-known example with different demand multipliers.

```
... bmult = mi.sync_db.add_parameter("bmult", 0, "demand multiplier")
opt = ws.add_options()
opt.all_model_types = "gurobi"
mi.instantiate("transport use lp min z", GamsModifier(bmult), opt)
bmult.add_record().value = 1.0
bmultlist = [ 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 ]
for b in bmultlist:
    bmult.first_record().value = b
    print "Scenario bmult=" + str(b) + ":" + str(mi.model_status)
    print "Solvestatus: " + str(mi.solver_status)
    print "Obj: " + str(mi.sync_db.get_variable("z").find_record().level)
...```

7.11.3.20 How to modify a variable of a GamsModelInstance using GamsModifier (transport7.py)

We create a GamsModelInstance using the GamsCheckpoint.add_modelinstance method. Afterwards we define x as GamsVariable and a GamsParameter xup that will be used as upper bound for x. At the following instantiate method GamsModifier has 3 arguments. The first one says that x is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GamsParameter that holds the new value, in this case xup.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
... mi = cp.add_modelinstance()
x = mi.sync_db.add_variable("x", 2, VarType.Positive)
xup = mi.sync_db.add_parameter("xup", 2, "upper bound on x")
mi.instantiate("transport use lp min z", GamsModifier(x, UpdateAction.Upper, xup))
mi.solve()
for i in t7.out_db["i"]:  
    for j in t7.out_db["j"]: 
        xup.clear()
        xup.add_record((i.keys[0],j.keys[0])).value = 0
        mi.solve()
        print "Scenario link blocked: " + i.keys[0] + " - " + j.keys[0]
        print "Modelstatus: " + str(mi.model_status)
        print "Solvestatus: " + str(mi.solver_status)
        print "Obj: " + str(mi.sync_db["z"].find_record().level)
...```
7.11.3.21 How to use a queue to solve multiple GamsModelInstances in parallel (transport8.py)

We initialize a GamsCheckpoint cp from a GamsJob. Then we define a list that represents the different values of the demand multiplier. That list will be used like a queue where we extract the last element first. The objects list_lock and io_lock are used later to avoid multiple reading of one demand multiplier and messed up output. Then we call function scen_solve multiple times in parallel. The number of parallel calls is specified by nr_workers.

In function scen_solve we create and instantiate a GamsModelInstance as in the preceding examples and make parameter bmult modifiable. Note that we choose cplex as solver because it is thread safe (gurobi would also be possible).

We have two critical sections that are locked by the objects list_lock and io_lock. Note that the pop method removes and returns the last element from the list and deletes it. Once the list is empty the loop terminates.

7.11.3.22 How to fill a GamsDatabase by reading from MS Access (transport9.py)

This example illustrates how to import data from Microsoft Access to a GamsDatabase. There are a few prerequisites required to run transport9.py successfully.

- We import pyodbc which can be downloaded from [http://code.google.com/p/pyodbc/](http://code.google.com/p/pyodbc/).
- Note that an architecture mismatch might cause problems. The bitness of your MS Access, Python, pyodbc and GAMS should be identical.
We call a function *read_data_from_access* that finally returns a GamsDatabase as shown below.

```python
    db = read_from_access(ws)
```

The function *read_from_access* begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database *transport.accdb* which can be found in `\[GAMSDIR]\apifiles\Data`. To finally read in GAMS sets and parameters we call the functions *read_set* and *read_parameter* that are explained down below.

```python
def read_from_access(ws):
    db = ws.add_database()
    # connect to database
    str_access_conn = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=..\\Data\transport.accdb'
    try:
        connection = pyodbc.connect(str_access_conn)
    except Exception as ex:
        print "Error: Failed to create a database connection.\n{n [0]}\format(ex)"
        sys.exit(1)
    # read GAMS sets
    read_set(connection, db, "SELECT Plant FROM plant", "i", 1, "canning plants")
    read_set(connection, db, "SELECT Market FROM Market", "j", 1, "markets")
    # read GAMS parameters
    read_parameter(connection, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases")
    read_parameter(connection, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases")
    read_parameter(connection, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles")
    connection.close()
    return db
```

The function *read_set* adds a set to the GamsDatabase that is filled with the data from the MS Access file afterwards. The function *read_parameter* works quite similar.

```python
def read_set(connection, db, query_string, set_name, set_dim, set_exp=""):
    try:
        cursor = connection.cursor()
        cursor.execute(query_string)
        data = cursor.fetchall()
        if len(data[0]) != set_dim:
            print "Number of fields in select statement does not match setDim"
            sys.exit(1)
        i = db.add_set(set_name, set_dim, set_exp)
        for row in data:
            keys = []
            for key in row:
                keys.append(str(key))
            i.add_record(keys)
    except Exception as ex:
        print "Error: Failed to retrieve the required data from the DataBase.\n{n [0]}\format(ex)"
        sys.exit(1)
    finally:
        cursor.close()
```

Once we read in all the data we can create a GamsJob from the GamsDatabase and run it as usual.

**7.11.3.23 How to fill a GamsDatabase by reading from MS Excel (transport10.py)**

This example illustrates how to read data from Excel, or to be more specific, from `\[GAMSDIR]\apifiles\Data\transport.xlsx`.

At first you have to download an additional package from pypi, e.g. to `C:\[USERNAME]\Downloads`. Unzip the package and run the following code from command line:

```bash
    cd C:\[Username]\Downloads\xlrd-0.9.2 && python setup.py install && cd \[GAMSDIR]\apifiles\Python
```

Now you should be able to find the directory `xlrd` in `\[PYTHONDIR]\Lib\site-packages` and to run `transport10.py`.

In `transport10.py` the model is given as string without data like in many examples before and the Excel file `transport.xlsx` is located at `\[GAMSDIR]\apifiles\Data`. At first we define the workbook to read from and the different sheet names. To ensure to have the same number of markets and plants in all
spreadsheets, we conduct a little test that checks for the number of rows and columns. Our workspace is only created if this test yields no errors.

```python
wb = open_workbook("..\Data\transport.xlsx")
capacity = wb.sheet_by_name("capacity")
demand = wb.sheet_by_name("demand")
distance = wb.sheet_by_name("distance")
assert (distance.ncols-1 == demand.ncols) and (distance.nrows-1 == capacity.ncols), "Size of the spreadsheets doesn’t match"
```

Now we can create a GamsDatabase and read in the data contained in the different worksheets. We iterate over the columns and read in the set names and the corresponding parameter values.

```python
db = ws.add_database()
i = db.add_set("i", 1, "Plants")
j = db.add_set("j", 1, "Markets")
capacity_param = db.add_parameter("a", 1, "Capacity")
demand_param = db.add_parameter("b", 1, "Demand")
distance_param = db.add_parameter("d", 2, "Distance")
for cx in range(capacity.ncols):
    capacity_param.add_record(str(capacity.cell_value(rowx=0, colx=cx))).value = capacity.cell_value(rowx=1, colx=cx)
for cx in range(demand.ncols):
    demand_param.add_record(str(demand.cell_value(rowx=0, colx=cx))).value = demand.cell_value(rowx=1, colx=cx)
for cx in range(1, distance.ncols):
    for rx in range(1, distance.nrows):
        keys = (str(distance.cell_value(rowx=rx, colx=0)), str(distance.cell_value(rowx=0, colx=cx)))
        distance_param.add_record(keys).value = distance.cell_value(rowx=rx, colx=cx)
```

Note that we can name sets and parameters just like in the database but we don’t have to. Now we can run our GamsJob as usual.

```python
t10 = ws.add_job_from_string(get_model_text())
opt = ws.add_options()
opt.defined["gdxincname"] = db.name
for rec in t10.out_db["x"]:  
    print "x(" + rec.keys[0] + ", " + rec.keys[1] + "): level=" + str(rec.level) + " marginal=" + str(rec.marginal) 
```

### 7.11.3.24 How to create and use a save/restart file (transport11.py)

In transport11.py we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a GamsCheckpoint.

We create a directory `tmp` with internal identifier `w_dir` in the directory we are currently in (usually `[GAMSDIR]\apifiles\Python`). This file will be used as working directory later. From the main function we call the function `create_save_restart` giving it directory `tmp` and the desired name for the save/restart file (`tbase`) as arguments.

```python
w_dir = os.path.join(".", "tmp")
create_save_restart(os.path.join(w_dir, "tbase"));
```

In function `create_save_restart` we create a workspace with the given working directory (`w_dir` refers to `tmp`). Then we create a GamsJob from a string. Note that the string given via `get_base_model_text` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a GamsOption to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the GamsJob and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the `add_checkpoint` method accepts identifiers as well as file names as argument.

```python
def create_save_restart(cp_file_name):
    if len(sys.argv) > 1:
        ws = GamsWorkspace(os.path.dirname(cp_file_name), sys.argv[1])
    else:
        ws = GamsWorkspace(os.path.dirname(cp_file_name))
```
ws = GamsWorkspace(os.path.dirname(cp_file_name))

j1 = ws.add_job_from_string(get_base_model_text())
opt = ws.add_options()
opt.action = Action.CompileOnly
cp = ws.add_checkpoint(os.path.basename(cp_file_name))
j1.run(opt, cp)
del opt

So what you should keep in mind before we return to further explanations of the main function is, that
the file tbase is now in the current working directory and contains a checkpoint that will work exactly
like a restart file.

In the main function we define some data using Python data structures as we already did in transport4.py
before we create the GamsWorkspace and a GamsDatabase.

if len(sys.argv) > 1:
    ws = GamsWorkspace(w_dir, sys.argv[1])
else:
    ws = GamsWorkspace(w_dir)

Afterwards we set up the GamsDatabase like we already did in transport4.py. Once this is done we run a
GamsJob using this data plus the checkpoint stored in file tbase.

Note that the string from which we create job t11 is different to the one used to prepare the checkpoint
stored in tbase and is only responsible for reading in the data from the GamsDatabase correctly. The
entire model definition is delivered by the checkpoint cp_base which is equal to the one we saved in tbase.

7.12 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Matlab
API. It allows the user to start immediately working with the API by providing a set of small examples
based on the well-known transportation problem. These examples introduce several API features step
by step.

- Getting started A quick introduction about how to add the API to the Matlab path
- Important Classes of the API Overview of some fundamental classes of the GAMS Matlab API
- How to use the API An extensive set of examples describing how to use API components

7.12.1 Getting started

The object oriented GAMS Matlab API is built on top of the GAMS Java API and provides convenient
access to GAMS from within Matlab (2017b or later) and Octave (5.2 or later). Examples using the
API are located in apifiles/Matlab/examples and apifiles/Matlab/examples_octave for Matlab or
Octave, respectively. The examples slightly differ for Octave since it does not support the complete Matlab
syntax and functionalities - most importantly to note, enumeration classes (see also the note below). The
object oriented API is located in apifiles/Matlab/api.

For making your Matlab installation aware of the packages provided by GAMS, you need to do the
following two steps. First, add the GAMS Java API to the Matlab java class path. This can be done
statically or dynamically. The latter is less efficient but more convenient for testing.
• static: Locate the javaclasspath.txt and add the path [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar. You can find the location with cd(prefdir) in Matlab or which javaclasspath in Octave. Finally, restart Matlab or Octave.

• dynamic: Call the function javaaddpath("[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar").

Second, add the GAMS Matlab API to the Matlab path by calling addpath("[PathToGAMS]/apifiles/Matlab/api").

Note

Octave does not support enumeration classes (yet). In order to use the GAMS enumerations, please use the alternative integer or string value. For example, instead of using GAMS.DebugLevel.SHOW_LOG, use 2, see DebugLevel. To tell the GAMS Matlab API to return the alternative values instead of instances of enumeration classes, set the environment variable GAMS_MATLAB_ENUM_ALT to true, i.e., setenv("GAMS_MATLAB_ENUM_ALT", "true");.

You can test your setup by creating a GAMSWorkspaceInfo:
>> wsinfo = GAMS.GAMSWorkspaceInfo()
wsinfo =
    GAMSWorkspaceInfo with properties:
        debugLevel: OFF
        systemDirectory: ''
        workingDirectory: ''

Note

In Matlab you can import the GAMS package by import GAMS. ∗ Then, you don't need to call the GAMS classes with the preceding GAMS..

### 7.12.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS package. Their usage is demonstrated by an extensive set of examples. All GAMS Matlab API classes are contained within one single package GAMS. It provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this package allow convenient exchange of input data and model results (GAMSDatabase) and help to create and run GAMS models (GAMSJob), that can be customized by GAMS options (GAMSOptions). Furthermore, it introduces a way to solve a sequence of closely related models in the most efficient way (GAMSMModelInstance).

Other classes are GAMSWorkspace, GAMSOptions and GAMSSymbol.

### 7.12.3 How to use the API

In the GAMS system directory there are some examples provided that illustrate the usage of the Matlab API. [PathToGAMS]/apifiles/Matlab/examples contains multiple examples dealing - among others - with the well-known transportation problem. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

• How to choose the GAMS system (transport1)
• How to export data to GDX (transport_gdx)
• How to import data from GDX (transport_gdx)
• How to run a GAMSJob from file (transport1)
• How to retrieve a solution from an output database (transport1)
• How to specify the solver using GAMSOptions (transport1)
• How to run a job with a solver option file and capture its log output (transport1)
• How to use include files (transport2)
• How to set a non-default working directory (transport3)
• How to read data from string and export to GDX (transport3)
• How to run a job using data from GDX (transport3)
• How to run a job using implicit database communication (transport3)
• How to define data using Matlab data structures (transport4)
• How to prepare a GAMSDatabase from Matlab data structures (transport4)
• How to initialize a GAMSCheckpoint by running a GAMSJob (transport5)
• How to initialize a GAMSJob from a GAMSCheckpoint (transport5)
• How to create a GAMSModelInstance from a GAMSCheckpoint (transport7)
• How to modify a parameter of a GAMSModelInstance using GAMSModifier (transport7)
• How to modify a variable of a GAMSModelInstance using GAMSModifier (transport7)
• How to create and use a save/restart file (transport11)

### 7.12.3.1 How to choose the GAMS system (transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on
your machine, the desired system can be specified via an additional argument when the workspace is created.
When running the examples, we can provide an additional command line argument in order to define the
GAMS system directory that should be used. By executing `transport1` with `C:/GAMS/win64/40.1` we
use the 64-bit version of GAMS 40.1 to run `transport1` even if our default GAMS system might be a
different one. This is managed by the following code:

```matlab
wsInfo = GAMS.GAMSWorkspaceInfo();
if nargin > 0
    wsInfo.systemDirectory = varargin{1};
end
ws = GAMS.GAMSWorkspace(wsInfo);
```

Note

The API can detect GAMS automatically from the *PATH* environment variable. Please note that
this is not the *MATLABPATH*. You can inspect the *PATH* with `getenv("PATH")`. 
7.12.3.2 How to export data to GDX (transport.gdx)

Although the Object-oriented Matlab API offers much more than exchanging data between Matlab and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in Matlab data structures.

\begin{verbatim}
plants = {'Seattle', 'San-Diego'};
markets = {'New-York', 'Chicago', 'Topeka'};
capacity = containers.Map();
capacity('Seattle') = 350;
capacity('San-Diego') = 600;
demand = containers.Map();
demand('New-York') = 325;
demand('Chicago') = 300;
demand('Topeka') = 275;
distance = containers.Map();
distance('Seattle.New-York') = 2.5;
distance('Seattle.Chicago') = 1.7;
distance('Seattle.Topeka') = 1.8;
distance('San-Diego.New-York') = 2.5;
distance('San-Diego.Chicago') = 1.8;
distance('San-Diego.Topeka') = 1.4;
\end{verbatim}

Different type of GAMS symbols are represented using different Matlab data structures. The data for the GAMS sets is represented using a cell of strings (e.g. plants and markets). On the other hand, GAMS parameters are represented by a containers.Map (e.g. capacity and demand). Note that the representation of the two dimensional parameter distance uses a dot notation for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with Matlab data structures.

A new GAMSDatabase instance can be created using GAMSWorkspace.addDatabase.

\begin{verbatim}
db = ws.addDatabase();
\end{verbatim}

We start adding GAMS sets using the method GAMSDatabase.addSet which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through plants and adds new records to the recently created GAMSSet instance \texttt{i} using GAMSSet.addRecord.

\begin{verbatim}
i = db.addSet('i', 1, 'canning plants');
for p = plants
  i.addRecord(p{1});
end
\end{verbatim}

GAMSParameter instances can be added by using the method GAMSDatabase.addParameter. In this example we use the overloaded method which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

\begin{verbatim}
a = db.addParameter('a', 'capacity of plant i in cases', i);
for p = plants
  a.addRecord(p{1}).value = capacity(p{1});
end
\end{verbatim}

As soon as all data is prepared in the GAMSDatabase, the method GAMSDatabase.export can be used to create a GDX file.

\begin{verbatim}
db.export('data.gdx');
\end{verbatim}

7.12.3.3 How to import data from GDX (transport.gdx)

Data can be imported from a GDX file using GAMSWorkspace.addDatabaseFromGDX. The method takes a path to a GDX file and creates a GAMSDatabase instance.

\begin{verbatim}
gdxdb = ws.addDatabaseFromGDX('data.gdx');
\end{verbatim}

Reading the data from the GAMSSet \texttt{i} into a cell of strings can be done as follows:

\begin{verbatim}
gdxPlantsRecords = gdxdb.getSet('i').records;
gdxPlants = cell(size(gdxPlantsRecords));
for i = 1:numel(gdxPlantsRecords)
  gdxPlants{i} = gdxPlantsRecords{i}.key(1);
end
\end{verbatim}

\texttt{i} is retrieved by calling GAMSDatabase.getSet on gdxdb. The returned GAMSSet object has an attribute records with an cell array of GAMSSetRecords. set. Each record can be asked for its keys.

\begin{verbatim}
\end{verbatim}
7.12 Tutorial

You can do the same for GAMSParameter. Instead of creating a cell, we want to have the data in the form of a containers.Map. GAMSParameterRecord can not only be asked for its keys, but also for its value. The following code snippet shows how to read the one dimensional parameter \( a \) into a map.

```plaintext
gdxCapacity = containers.Map();
for rec = gdxdb.getParameter('a').records
    gdxCapacity(rec[1].key(1)) = rec[1].value;
end
```

For a key of multi dimensional symbol, we choose a dot based concatenation of keys.

```plaintext
gdxDistance = containers.Map();
for rec = gdxdb.getParameter('d').records
    gdxDistance([rec[1].key(1), '.', rec[1].key(2)]) = rec[1].value;
end
```

Scalar can be read into a variable of type double by accessing the value of the first and only record.

```plaintext
gdxFreight = gdxdb.getParameter('f').record.value;
```

7.12.3.4 How to run a GAMSJob from file (transport1)

At first we create our workspace using `GAMSWorkspace`:

```plaintext
ws = GAMS.GAMSWorkspace();
```

Afterwards, we can create a `GAMSJob t1` using the `addJobFromGamsLib` method and run it.

```plaintext
% create GAMSWorkspace "ws" with default working directory
ws = GAMS.GAMSWorkspace();
% create GAMSJob "t1" from "trnsport" model in GAMS Model Libraries
    t1 = ws.addJobFromGamsLib('trnsport');
% run GAMSJob "t1"
    t1.run();
```

7.12.3.5 How to retrieve a solution from an output database (transport1)

The following lines create the solution output and illustrate the usage of the `GAMSJob.outDB` property to get access to the `GAMSDatabase` created by the `run` method. To retrieve the content of variable \( x \) we use the `getVariable` method and the `GAMSVariableRecord` class.

```plaintext
% retrieve GAMSVariable "x" from GAMSJob's output databases
fprintf('Ran with Default:

for x = t1.outDB.getVariable('x').records
    fprintf('x(%s,%s): level=%g marginal=%g

end
```

7.12.3.6 How to specify the solver using GAMSOptions (transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.addOptions` method. The `GAMSOptions.setAllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle. Then we run our `GAMSJob t1` with the new `GAMSOptions`.

```plaintext
% create GAMSOptions 'opt1'
    opt1 = ws.addOptions();
% set all model types of 'opt1' for 'xpress'
    opt1.setAllModelTypes('xpress');
% run GAMSJob 't1' with GAMSOptions 'opt1'
    t1.run(opt1);
```
7.12.3.7 How to run a job with a solver option file and capture its log output (transport1)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterward we use a `GAMSOptions` just like in the preceding example and `GAMSOptions.optFile` property to 1 to tell the solver to look for a solver option file. We specify the argument `output` in order to stream the log of the GAMSJob into the file `transport1_xpress.log`. When the `output` argument is omitted then the log will be written to standard output.

```plaintext
% write file 'xpress.opt' under GAMSWorkspace's working directory
fid = fopen([ws.workingDirectory, GAMS.GAMSGlobals.FILE_SEPARATOR, 'xpress.opt'], 'w');
fprintf(fid, 'algorithm=barrier');
fclose(fid);
%
% create GAMSOptions 'opt2'
% set all model types of 'opt2' for 'xpress'
% for 'opt2', use 'xpress.opt' as solver's option file
% run GAMSJob 't2' with GAMSOptions 'opt2' and capture log into 'transport1_xpress.log'.
t1.run(opt2, [ws.workingDirectory, GAMS.GAMSGlobals.FILE_SEPARATOR, 'transport1_xpress.log']);
```

7.12.3.8 How to use include files (transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings `data` and `model` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```plaintext
% write 'data' into file 'tdata.gms' under GAMSWorkspace's working directory
fid = fopen([ws.workingDirectory, GAMS.GAMSGlobals.FILE_SEPARATOR, 'tdata.gms'], 'w');
fprintf(fid, data);
fclose(fid);
%
% create GAMSJob 't2' from the 'model' string variable
% define 'incname' as 'tdata'
% run GAMSJob 't2' with GAMSOptions 'opt2' and capture log into 'transport1_xpress.log'.
t2.run(opt2, [ws.workingDirectory, GAMS.GAMSGlobals.FILE_SEPARATOR, 'transport1_xpress.log']);
```

7.12.3.9 How to set a non-default working directory (transport3)

At first we create a new directory. Once this is done we can use this directory when creating the `GAMSWorkspace` and make it the working directory.

```plaintext
% create a directory
workingDirectory = fullfile(pwd, 'transport3');
mkdir(workingDirectory);
%
% create a workspace
wsInfo = GAMS.GAMSworkspaceInfo();
wsInfo.workingDirectory = workingDirectory;
ws = GAMS.GAMSworkspace(wsInfo);
```

7.12.3.10 How to read data from string and export to GDX (transport3)

We read the data from the string data. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding `GAMSJob` a `GAMSDatabase` is created that is available via the `GAMSJob.outDB` property. We can use the `GAMSDatabase.export` method to write the content of this database to a gdx file `tdata.gdx`.

```plaintext
% create and run a job from a data file, then explicitly export to a GDX file
% write file 'tdata.gdx' under GAMSWorkspace's working directory
% create GAMSOption 'opt' and define 'incname' as 'tdata'
% run GAMSJob 't2' with GAMSOptions 'opt' and capture log into 'transport1_xpress.log'.
t3.run();
t3.outDB.export([ws.workingDirectory, GAMS.GAMSGlobals.FILE_SEPARATOR, 'tdata.gdx']);
```
7.12.3.11 How to run a job using data from GDX (transport3)

This works quite similar to the usage of an include file explained in transport2 - How to use include files (transport2).

```gams
% run a job using an instance of GAMSOptions that defines the data include file

t3 = ws.addJobFromString(model);
opt = ws.addOptions();
opt.defines('gdxincname', 'tdata');
t3.run(opt);
```

Note that there are some minor changes in the model due to the usage of a gdx instead of an include file.

```gams
if not set gdxincname
  abort 'no include file name for data file provided'

gdxin %gdxincname%
%load i j a b d f%
```

7.12.3.12 How to run a job using implicit database communication (transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs t3a and t3b where the first one contains only the data and the second one contains only the model without data. After running t3a the corresponding outDB can be read in directly just like a gdx file.

```gams
t3a = ws.addJobFromString(data);
t3b = ws.addJobFromString(model);
opt = ws.addOptions();
opt.defines('gdxincname', t3a.outDB.name);
t3b.run(opt, t3a.outDB);
```

7.12.3.13 How to define data using Matlab data structures (transport4)

We use cell arrays and containers.Map to define Matlab data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```matlab
plants = {'Seattle', 'San-Diego'};
markets = {'New-York', 'Chicago', 'Topeka'};
capacity = containers.Map();
capacity('Seattle') = 350;
capacity('San-Diego') = 600;
demand = containers.Map();
demand('New-York') = 325;
demand('Chicago') = 300;
demand('Topeka') = 275;
distance = containers.Map();
distance('Seattle.New-York') = 2.5;
distance('Seattle.Chicago') = 1.7;
distance('Seattle.Topeka') = 1.8;
distance('San-Diego.New-York') = 2.5;
distance('San-Diego.Chicago') = 1.8;
distance('San-Diego.Topeka') = 1.4;
```

7.12.3.14 How to prepare a GAMSDatabase from Matlab data structures (transport4)

At first we create an empty GAMSDatabase db using the GAMSWorkspace.addDatabase method.

```gams
GAMSJob can be run like explained in the preceding example about implicit database communication.
```
db = ws.addDatabase();
i = db.addSet('i', 1, 'canning plants');
for p = plants
    i.addRecord(p[1]);
end
j = db.addSet('j', 1, 'markets');
for m = markets
    j.addRecord(m[1]);
end
a = db.addParameter('a', 'capacity of plant i in cases', i);
for p = plants
    a.addRecord(p[1]).value = capacity(p[1]);
end
b = db.addParameter('b', 'demand at market j in cases', j);
for m = markets
    b.addRecord(m[1]).value = demand(m[1]);
end
d = db.addParameter('d', 'distance in thousands of miles', i, j);
for p = plants
    for m = markets
        d.addRecord(p[1], m[1]).value = distance([p[1], m[1]]);
    end
end
f = db.addParameter('f', 'freight in dollars per case per thousand miles');
f.addRecord().value = 90;

% create and run a job from the model and read gdx include file from the database

# 7.12.3.15 How to initialize a GAMSCheckpoint by running a GAMSJob (transport5)

The following lines of code conduct several operations. While the first line simply creates a GAMSCheckpoint, the second one uses the GAMSWorkspace.addJobFromString method to create a GAMSJob containing the model text and data but no solve statement. Afterwards the run method gets the GAMSCheckpoint as argument. That means the GAMSCheckpoint cp captures the state of the GAMSJob.

cp = ws.addCheckpoint();
t5 = ws.addJobFromString(model);
t5.run(cp);

# 7.12.3.16 How to initialize a GAMSJob from a GAMSCheckpoint (transport5)

Note that the string returned from function model contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

Scalar bmult demand multiplier /1/;
demand(j) .. sum(i, x(i,j)) =G= bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';

In transport5 we create a list with eight different values for this demand multiplier.
bmultlist = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3];

For each entry of that list we create a GAMSJob t5 using the GAMSWorkspace.addJobFromString method. Besides another string which resets the demand multiplier bmult, specifies the solve statement and assigns values to the scalars ms and ss we pass the checkpoint cp as additional argument. This results in a GAMSJob combined from the checkpoint plus the content provided by the string. We run the GAMSJob and echo some interesting data from the outDB using the GAMSDatabase.getParameter and GAMSDatabase.getVariable methods and the GAMSSymbol.record attribute plus the GAMSParameterRecord.value and the GAMSVariableRecord.level properties.

for i = 1:numel(bmultlist)
    job = sprintf('bmult=%f; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;', bmultlist(i));
t5 = ws.addJobFromString(job, cp);
t5.run();
fprintf('Scenario bmult=%f:
', bmultlist(i));
end
Note

Some of the demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.12.3.17 How to create a GAMSModelInstance from a GAMSCheckpoint (transport7)

In transport7 the usage of Matlab::GAMS::GAMSModelInstance is demonstrated.

At first checkpoint cp is created as in the preceding examples. Then we create the GAMSModelInstance mi using the GAMSCheckpoint.addModelInstance method. Note that the GAMSJob again contains no solve statement and the demand multiplier is already included with default value 1.

```gams
% initialize a checkpoint by running a job
t7 = ws.addJobFromString(model);
t7.run(cp);
% create a ModelInstance and solve it multiple times with different scalar bmult
mi = cp.addModelInstance();
```

7.12.3.18 How to modify a parameter of a GAMSModelInstance using GAMSModifier (transport7)

A GAMSModelInstance uses a syncDB to maintain the data. We define bmult as GAMSParameter using the GAMSParameter method and specify cplex as solver. Afterwards the GAMSModelInstance is instantiated with 3 arguments, the solve statement, GAMSOptions opt and GAMSModifier bmult. The GAMSModifier means that bmult is modifiable while all other parameters, variables and equations of ModelInstance mi stay unchanged. We use the GAMSParameter.addRecord method and the value property to assign a value to bmult. That value can be varied afterwards using the GAMSSymbol.record property to reproduce our well-known example with different demand multipliers.

```gams
bmult = mi.syncDB.addParameter('bmult', 'demand multiplier');
opt = ws.addOptions();
opt.setAllModelTypes('cplex');
% instantiate the ModelInstance and pass a model definition and Modifier to declare bmult mutable
mi.instantiate('transport use lp min z', opt, GAMS.GAMSModifier(bmult));
bmult.addRecord().value = 1.0;
bmultlist = [0.6, 0.7 , 0.8, 0.9, 1.0, 1.1, 1.2, 1.3];
for i = 1:numel(bmultlist)
    bmult.record.value = bmultlist(i);
    mi.solve();
    fprintf('Scenario bmult=%f: 
', bmultlist(i));
    fprintf(' Modelstatus: %s 
', mi.modelStatus);
    fprintf(' Solvestatus: %s 
', mi.solveStatus);
    fprintf(' Obj: %f 
', mi.syncDB.getVariable('z').record.level);
end
```

7.12.3.19 How to modify a variable of a GAMSModelInstance using GAMSModifier (transport7)

We create a GAMSModelInstance just like in the next to last example. We define x as GAMSVariable and its upper bound as GAMSParameter xup. At the following instantiate method GAMSModifier has 3 arguments. The first one says that x is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GAMSParameter that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```gams
x = mi.syncDB.addVariable('x', 2, GAMS.VarType.POSITIVE, '');
xup = mi.syncDB.addParameter('xup', 2, 'upper bound on x');
% instantiate the ModelInstance and pass a model definition and Modifier to declare upper bound of X mutable
mi.instantiate('transport use lp min z', GAMS.GAMSModifier(x, GAMS.UpdateAction.UPPER, xup));
```
for i = t7.outDB.getSet('i').records
    for j = t7.outDB.getSet('j').records
        xup.clear();
        keys = [{i[1].key(1), j[1].key(1)};
        xup.addRecord(keys).value = 0;
        mi.solve();
        fprintf('Scenario link blocked: %s - %s
        ', keys);
        fprintf(' Modelstatus: %s
        ', mi.modelStatus);
        fprintf(' Solvestatus: %s
        ', mi.solveStatus);
        fprintf(' Obj: %f
        ', mi.syncDB.getVariable('z').record.level);
    end
end

7.12.3.20 How to create and use a save/restart file (transport11)

In transport11 we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a GAMSCheckpoint. From the main function we call the function `create_save_restart` giving it the current workspace settings and the desired file name as arguments.

create_save_restart(ws.workingDirectory, ws.systemDirectory, ws.debugLevel, 'tbase');

In function `create_save_restart` we create a workspace with the given workspace settings. Then we create a GAMSJob from a string. Note that the string given via `basemodel` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a GAMSOptions to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the GAMSJob and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the `addCheckpoint` method accepts identifiers as well as file names as argument.

ws = GAMS.GAMSWorkspace(workdir, systemdir, debuglevel);
j1 = ws.addJobFromString(basemodel);
opt = ws.addOptions();
opt.action = GAMS.Action.CompileOnly;
cp = ws.addCheckpoint('tbase');
j1.run(opt, cp);
opt.dispose();
j1.outDB.dispose();

So what you should keep in mind before we return to further explanations of the main function is, that the file `tbase` is now in the current working directory and contains a checkpoint. Now in the main function we define some data using Matlab data structures as we already did in transport4 before we create the GAMSWorkspace and a GAMSDatabase.

ws = GAMS.GAMSWorkspace(wsInfo);
db = ws.addDatabase();

Afterwards we set up the GAMSDatabase like we already did in transport4. Once this is done we run a GAMSJob using this data plus the checkpoint stored in file tbase.

cpBase = ws.addCheckpoint('tbase');
opt = ws.addOptions();
t11 = ws.addJobFromString(model, cpBase);
opt.defines('gdxincname', db.name);
opt.setAllModelTypes('xpress');
t11.run(opt, db);

Note that the string from which we create job `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the GAMSDatabase correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

7.13 Release Notes

7.13.1 39.1.0

- Added GAMSSet.SetType read property to distinguish multi and singleton sets.
- New optional argument setType=SetType.multi in GAMSDatabase.AddSet().
7.13.2 35.1.0

- Fixed `GAMSOptions.Output` parameter being ignored in `GAMSJob.Run()`.

7.13.3 32.1.0

- Automatically finding a GAMS installation from the Windows registry has been changed from using `[HKEY_CURRENT_USER\HKEY_CLASSES_ROOT]\gams.location` to `[HKEY_CURRENT_USER\HKEY_LOCAL_MACHINE]\Software`.

7.13.4 25.0.1

Changes for `GAMSSymbol` and `GAMSSymbolRecord`: Both classes got the `IEquatable<T>` Interface. As a result the behavior of the `Equals` function as well as the `==` and `!=` operators were modified. For both classes `Equals` and `==` returns now true, if the internal data reference is the same. Here is an example:

```csharp
GAMSVariable x1 = t1.OutDB.GetVariable("x");
GAMSVariable x2 = t1.OutDB.GetVariable("x");
if(x1 == x2)
    Console.WriteLine("x1==x2");
else
    Console.WriteLine("x1!=x2");
if (x1.Equals(x2))
    Console.WriteLine("x1 equals x2");
else
    Console.WriteLine("x1 does not equal x2");
```

In previous versions we got this output:

```
x1!=x2
x1 does not equal x2
```

Now we get:

```
x1==x2
x1 equals x2
```

7.13.5 24.9.1

- New functions `GAMSWorkspace.AddJobFromApiLib`, `GAMSWorkspace.AddJobFromNoaLib` to create `GAMSJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library

7.13.6 24.8.1

- New option `GAMSOptions.ErrorLog`: Max error message lines written to the log for each error

7.13.7 24.7.4

- `GAMSModelInstance.Instantiate`: Skip creation of GDX file, which was unreachable from within the API anyway

7.13.8 24.7.1

- `GAMSWorkspace.AddJobFromFile`: Throw an exception if given file does not exist
7.13.9 24.5.4

- Fixed a bug with the property GAMSOptions.IDir

7.13.10 24.5.2

- Fixed a bug that lead to a crash on Linux (using Mono) when the GAMS system directory is a symbolic link

7.13.11 24.5.1

- New functions GAMSWorkspace.ApiLib and GAMSWorkspace.NoaLib that can be used to retrieve models from the GAMS API Library and the Nonlinear Optimization Applications Library

7.13.12 24.4.2

- New property GAMSSymbol.DomainsAsString: Domains of Symbol, each element is a string
  Note: If the domain is an alias in GAMS, this call will return the name of the Alias, not the name of the aliased Set
- Disable unwanted debug output from Couenne when running with GAMSModelInstance

7.13.13 24.4.1

- Modified handling of GAMS Aliases:
  - If we ask for the number of GAMSSymbols in a GAMSDatabase, the Aliases will be excluded
  - If we iterate over all GAMSSymbols in a GAMSDatabase, Aliases will be skipped
  - If we ask explicitly for an Alias in a GAMSDatabase (GAMSDatabase.GetSet(“a”) with a being an Alias) we will get a reference to the GAMSSet referenced by the Alias, not the Alias itself
  - Note: Aliases can appear in a GAMSDatabase only, if it was initialized by a GDX file containing an Alias.

7.13.14 24.3.2

- Make more GAMS options available through the GAMSOptions class:
  - GAMSOptions.AppendExpand: Expand file append option
  - GAMSOptions.AppendOut: Output file append option
  - GAMSOptions.DumpOpt: Writes preprocessed input to the file input.dmp
  - GAMSOptions.DumpParms: GAMS parameter logging
  - GAMSOptions.ErrMsg: Placing of compilation error messages
  - GAMSOptions.Expand: Expanded (include) input file name
  - GAMSOptions.FErr: Alternative error message file
  - GAMSOptions.JobTrace: Job trace string to be written to the trace file at the end of a Gams job
- GAMSOptions.LimCol: Maximum number of columns listed in one variable block
- GAMSOptions.LimRow: Maximum number of rows listed in one equation block
- GAMSOptions.LogLine: Amount of line tracing to the log file
- GAMSOptions.On115: Generate errors for unknown unique element in an equation
- GAMSOptions.Output: Output file
- GAMSOptions.PageContr: Output file page control option
- GAMSOptions.PageSize: Output file page size (=0 no paging)
- GAMSOptions.PageWidth: Output file page width
- GAMSOptions.Reference: Symbol reference file
- GAMSOptions.ScriptExit: Program or script to be executed at the end of a GAMS run
- GAMSOptions.Suppress: Compiler listing option
- GAMSOptions.Symbol: Symbol table file
- GAMSOptions.TraceLevel: Solvestat threshold used in conjunction with a=GT

### 7.13.15 24.3.1

- GAMSJob.Run creates OutDB now even if it throws a GAMSExceptionExecution
- New function GAMSSymbolRecord.Key(int index): Retrieve key of GAMSSymbolRecord on position index
- GAMSMModelInstance.Solve also works now for solvers which require a subsolver, e.g. Dicopt
- GAMSDatabase.Export registers real domains in GDX file now if possible, before relaxed domains were registered always
- GAMSSymbol.CopySymbol works for the Universe of a GAMSDatabase (GAMSDatabase.GetSet("*")) now, when copying into the Universe, a merge will be performed
- Fix default value for systemDirectory argument in GAMSWorkspace constructor when using MONO: If no value is given, first the PATH and then the (DY)LD_LIBRARY_PATH is checked for a valid GAMS system directory

### 7.13.16 24.2.3

- Fix memory leak in GAMSDatabase

### 7.13.17 24.2.2

- Fixed a bug regarding the subtype of equations and their default records, when the equation was added to a GamsDatabase from the API
- Fix GAMSEnum.GAMSExitCode
- Make sure that GAMSMModelInstance.Instantiate() does not solve the model, but only prepares everything required for the following Solve()
7.13.18  24.2.1

- New version of function GAMSWorkspace.AddOptions creating an instance of GAMSOptions initialized by an existing option file
- New properties: GAMSWorkspace.APIVersion, GAMSWorkspace.APIMajorRelNumber, GAMSWorkspace.APIMinorRelNumber and GAMSWorkspace.APIGOLDRelNumber
- New property GAMSWorkspace.MyEPS: Reset value to be stored in and read from GAMSDatabase for Epsilon, default is double.Epsilon
- New function GAMSOptions.Export: Write GAMSOptions into a parameter file
- Switching type of property GAMSOptions.NoNewVarEqu from Integer to Enum (ENoNewVarEqu)

7.13.19  24.1.1

- GAMSDatabase:
  - New function CheckDomains: Check for all symbols if all records are within the specified domain of the symbol
  - New function GetDatabaseDVs: Return all GAMSDatabaseDomainViolations
  - New property SuppressAutoDomainChecking: Controls whether domain checking is called in GAMSDatabase export
  - New variants of the functions AddEquation, AddParameter, AddSet, AddVariable: Allow to specify domain information
  - Function Compact becomes obsolete and will be dropped in future
- GAMSModelInstance:
  - New function CopyModelInstance: Copies a ModelInstance to a new ModelInstance which gets constructed at this call
  - New function Interrupt: Sends interrupt signal to running GAMSModelInstance
- GAMSModifier:
  - Allow to define SymbolUpdateType for each GAMSModifier separately
- GAMSSymbol (GAMSEquation, GAMSParameter, GAMSSet, GAMSVariable):
  - New function CheckDomains: Check if all records are within the specified domain of the symbol
  - New function CopyToArray: Copies values of a dense symbol into a dense array
  - New function CopyToSqzdArray: Copies values of a sparse symbol into a squeezed array
  - New function CopySparseToDenseArray: Copies values of a sparse symbol into a dense array
  - New function CopyFromDenseArray: Copies values from dense array into a symbol
  - New function CopySliceFromDenseArray: Copies values from slice of dense array into a symbol
  - New function GetSymbolDVs: Return all GAMSSymbolDomainViolations
  - New function MergeRecord: Finds record in GAMSSymbol if it exists, adds it if not
  - New property Domains: Domains of Symbol, each element is either a GAMSSet (real domain) or a string (relaxed domain)
- GAMSWorkspace:
  - Change the Debug parameter from a Boolean flag to an enum type called DebugLevel
7.13 Release Notes

- The Debug parameter can be overwritten by the system environment variable "GAMSOOAPI-DEBUG" if set to one of the following: Off, KeepFiles, ShowLog, Verbose
- Improve performance significantly for function AddDatabase(GAMSDatabase)
- New functions: AddJobFromGamsLib, AddJobFromTestLib, AddJobFromEmpLib, AddJobFromDataLib and AddJobFromFinLib
- New properties: Version, MajorRelNumber, MinorRelNumber and GOLDRelNumber
- New optional parameter inModelName for functions AddDatabaseFromGDX/AddDatabase: GAMS string constant that is used to access this database

- Add new sub class of GAMSEException: GAMSEExceptionExecution. This provides additional info about the reason of the failed execution.

7.13.20  24.0.2

- Add new function GAMSSymbol.CopySymbol which copies all records from one to another GAMSSymbol
- Label and symbol lookup is now case insensitive
- Fixed handling of infinite bounds for GAMSModifiers

7.13.21  24.0.1

- Add function Interrupt to GAMSJob
- Do not throw an exception when hitting an Alias with the GAMSDatabaseEnumerator or GetSymbol function but return the aliased GAMSSet

7.13.22  23.9.3

- Constructor of GAMSWorkspace now appends conditionally the GAMS system directory to the PATH environment variable. This fixes a problem with applications that create and dispose many GAMSWorkspace instances.

7.13.23  23.9.2

- GAMSModelInstance.Instantiate is now thread-safe
- Additional flag createOutDB for method GAMSJob.Solve which allows to switch off automatic outDB creation
- Fix a problem with parallel execution of GAMSModelInstances in Debug mode
- Add new class GAMSEException
- Add tighter restriction on adding symbols to GAMSModelInstance.SyncDB
- Add new class GAMSModelInstanceOpt to customize method GAMSModelInstance.Solve
  - GAMSModelInstanceOpt.Debug: produces a gams.gms, jacobian.gdx and dictmap.gdx allowing inspection of the model instance solved in the GAMSModelInstance.Solve method
  - GAMSModelInstanceOpt.NoMatchLimit: controls the maximum number of accepted unmatched scenario records before terminating the solve
  - GAMSModelInstanceOpt.OptFile: allows to choose solver option for GAMSModelInstance.Solve
  - GAMSModelInstanceOpt.Solver: allows to choose solver for GAMSModelInstance.Solve
- Add GAMSWorkspace.ScratchFilePrefix to prefix all scratch files created by GAMS.Net
- GAMSJob no longer implements IDisposable
7.14 Release Notes

7.14.1 39.1.0 (April 2022)

- Added `GAMSSet.getSetType` returning property to distinguish between multi and singleton sets.
- Added `GAMSDatabase.addSet` allowing to set type of Set.
- Fixed defined enumerated value for options `FDOpt` and `DumpOpt`.

7.14.2 35.1.0 (April 2021)

- Fixed `GAMSOptions.setOutput` parameter being ignored in `GAMSJob.run()`.

7.14.3 32.1.0 (July 2020)

- On windows platform, change of window registry configuration used for finding the installed GAMS directory from: `\[HKEY_CURRENT_USER\|HKEY_CLASSES_ROOT]\gams.location` to: `\[HKEY_CURRENT_USER\|HKEY_LOCAL_MACHINE]\Software\Classes\gams.location`, effective only when the installed GAMS directory is not found from "PATH" environment variable.

7.14.4 29.1.0 (November 2019)

- Increase the minimum version requirement of the Java Runtime Environment to Java SE 8.
- The remove operation `GAMSDatabaseIterator.remove` is no longer supported. The method now throws an instance of `java.lang.UnsupportedOperationException` and performs no other actions. (To remove all records of the current iterating symbol, use the method `GAMSSymbol.clear` instead.)
- Fixed the behavior of the remove operation `GAMSSymbolIterator.remove`. The method now removes all records of the last GAMSSymbol element returned by an instance of `GAMSSymbolIterator` and can be called only once per call to `GAMSSymbolIterator.next`. The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling the method.

7.14.5 28.2.0 (August 2019)

- Added new method `GAMSSymbolRecord.dispose` for on-demand release of external resources hold by non-java library.

7.14.6 28.1.0 (August 2019)

- Announced a plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.
7.14.7  25.1.1 (May 2018)

- Changed the naming scheme of a temporary working directory to be created from 'yyyyM-
  Mdd_HHmmss' to the prefixed 'gams-' (defined by `GAMSGlobalGlobals.workingDirectoryPrefix`),
in case no working directory has been specified.

- Fixed the behavior when GAMSDatabase is added with the name that already exists, now GAM-
  SException will be raised. (see `GAMSWorkspace.addDatabase(String databaseName)` and
  `GAMSWorkspace.addDatabaseFromGDX(String gdxFileName, String databaseName)`)  

- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available.
  As calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially
  create unpredictable outcome. Whenever the object is no longer needed it is recommended to
  explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See
  `GAMSGlobalDatabase.dispose`, `GAMSModelInstance.dispose`, and `GAMSOptions.dispose`.

- All Java native interfaces to Expert-Level APIs are now included in the distributed
  `GAMSJavaAPI.jar`, located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.

7.14.8  25.0.1 (January 2018)

- The minimum version requirement of Java Runtime Environment for using with GAMS Java API is
  now Java SE 7.

- New `TransportGDX` example to demonstrate how to import and export GDX files.

- Removed `GAMSSymbol.compact`, deprecated since 24.8.1 (December 2016).

- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two
  symbol objects with the same internal reference are now equivalent, similar to symbol record objects:

  - Two symbols are equivalent if and only if they have the same internal reference.
  - Two symbol records are equivalent if and only if they have the same internal reference.

  The behavior of operator `== remains unchanged. The following example illustrates the new equiva-
  lence behavior:

  ```java
  GAMSVariable x1 = db.getVariable("x");
  GAMSVariable x2 = db.getVariable("x");
  GAMSVariable x3 = x1;
  assertTrue(x1.equals(x2)); // true, previously false
  assertFalse(x1 == x2);  // false, previously false
  assertTrue(x1.equals(x3)); // true, previously true
  assertTrue(x1 == x3);   // true, previously true
  ```

7.14.9  24.8.1 (December 2016)

- Deprecated `GAMSSymbol.compact` and the method will be removed in the next major release

- an exception or an error thrown by GAMSWorkspace.finalize and GAMSSymbolIterator.finalize now
  must be caught or declared to be thrown. Note that an explicit call on one of these two methods is
  not recommended unless it is necessary to do so.

7.14.10  24.7.4 (September 2016)

- `GAMSModelInstance.instantiate`: Skip creation of GDX file, which was unreachable from
  within the API anyway.
7.14.11 24.7.1 (March 2016)

- Fixed a bug with the property `GAMSOptions.defines`: When too many entries were added, all of them were ignored.

7.14.12 24.5.1 (August 2015)

- New examples
  - `Clad` at `[Path/To/GAMS]/apifiles/Java/clad/Clad.java`
  - `SpecialValues` at `[Path/To/GAMS]/apifiles/Java/specialvalues/SpecialValues.java`

- New functions `GAMSWorkspace.addJobFromApiLib` and `GAMSWorkspace.addJobFromNoaLib` that can be used to retrieve models from GAMS API Library and Nonlinear Optimization Applications Library.

7.14.13 24.4.2 (March 2015)

- New property `GAMSSymbol.getDomainsAsStrings`: Get domains of symbol, each element is a string.
  Note: If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.

- Disable unwanted debug output from Couenne when running with `GAMSModelInstance`

- Change naming scheme of gdx output scratch file to sequence number.

7.14.14 24.4.1 (December 2014)

- Modify the handling of GAMS Aliases in the object oriented APIs:
  - If we ask for the number of `GAMSSymbol` in a `GAMSDatabase`, the Aliases will be excluded.
  - If we iterate over all `GAMSSymbol` in a `GAMSDatabase`, Aliases will be skipped.
  - If we ask explicitly for an Alias in a `GAMSDatabase` (`GAMSDatabase.getSet("a")` with `a` being an Alias) we will get a reference to the `GAMSSet` referenced by the Alias, not the Alias itself.
  - Note:
    * Aliases can appear in a `GAMSDatabase` only, if it was initialized by a GDX file containing an Alias.
    * The new example `Alias` at `[Path/To/GAMS]/apifiles/Java/alias/Alias.java` demonstrates this new behavior.

7.14.15 24.3.3 (September 2014)

Fixed

- a location of listing file when creating a job from (full-path) file without giving a job name
7.14.16 24.3.2 (August 2014)

- Make more GAMS options available through the GAMSOptions class:
  - GAMSOptions.EAppendExpand (enum, getter, and setter): Expand file append option
  - GAMSOptions.EAppendOut (enum, getter, and setter): Output file append option
  - GAMSOptions.EDumpOpt (enum, getter, and setter): Writes preprocessed input to the file input.dmp
  - GAMSOptions.EDumpParms (enum, getter, and setter): GAMS parameter logging
  - GAMSOptions.EAppendExpand (getter, and setter): Expanded (include) input file name
  - GAMSOptions.FErr (getter, and setter): Alternative error message file
  - GAMSOptions.JobTrace (getter, and setter): Job trace string to be written to the trace file at the end of a Gams job
  - GAMSOptions.LimCol (getter, and setter): Maximum number of columns listed in one variable block
  - GAMSOptions.LimRow (getter, and setter): Maximum number of rows listed in one equation block
  - GAMSOptions.ELogLine (enum, getter, and setter): Amount of line tracing to the log file
  - GAMSOptions.EOn115 (enum, getter, and setter): Generate errors for unknown unique element in an equation
  - GAMSOptions.Output (getter, and setter): Output file
  - GAMSOptions.EPageContr (enum, getter, and setter): Output file page control option
  - GAMSOptions.PageSize (getter, and setter): Output file page size (=0 no paging)
  - GAMSOptions.PageWidth (getter, and setter): Output file page width
  - GAMSOptions.Reference (getter, and setter): Symbol reference file
  - GAMSOptions.ScriptExit (getter, and setter): Program or script to be executed at the end of a GAMS run
  - GAMSOptions.ESuppress (enum, getter, and setter): Compiler listing option
  - GAMSOptions.Symbol (getter, and setter): Symbol table file
  - GAMSOptions.TraceLevel (getter, and setter): Solvestat threshold used in conjunction with a=GT

7.14.17 24.3.1 (July 2014)

New

- Transport14 example at [Path/To/GAMS]/apifiles/Java/transport/Transport14.java.
- GAMSSymbolRecord.getKey method: to retrieve key of GAMSWorkspace on specified position index.

Changed

- no longer necessary to set up environment variable before running a program
  - possible to specify the GAMS system directory during run time using GAMSWorkspaceInfo.setSystemDirectory(String directory) and GAMSWorkspace(GAMSWorkspaceInfo info)
• no longer necessary to specify `-Djava.library.path` when running a program
  – if `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`
  – otherwise the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
• when exporting a database to GDX: a symbol with real domains from now on will be registered
• in `GAMSJob.run`: always creates output database `OutDB` even if `GAMSExecutionException` has been raised.
• in `GAMSSymbol.copySymbol` when copy into the Universe symbol (`GAMSDatabase.getSet("*")`) of a `GAMSDatabase`: merge operation will be performed.
• in `GAMSWorkspace` default constructor: API will apply the default setting, that is, finding GAMS system directory from environment variable in the following order (depends on the target platform):
  – Windows: first from `PATH` environment variable. If not found, from the platform windows registry `gams.location`,
  – Mac OS: first from `PATH` environment variable. If not found, from `DYLD_LIBRARY_PATH`,
  – Unix: from `PATH` environment variable. If not found, from `LD_LIBRARY_PATH`.
• in non-default `GAMSWorkspace` constructor:
  – user can specify a GAMS system directory during run time, API will not search for a GAMS system directory from an environment variable.
  – in case the specified system directory is null or user specify other workspace attributes but a system directory, API will apply the default setting (see changed in `GAMSWorkspace` default constructor above).

Fixed

• API memory leak issue in `GAMSDatabaseIterator`.
• a bug when reading an option file on Unix platform with non-standard locale

Removed

• all deprecated classes and methods since 24.1.

7.14.18 24.2.3 (May 2014)

Fixed

• API memory leak issue in `GAMSDatabase`.
• a bug in `GAMSDatabase.getDatabaseDomainViolations`. 
7.14 Release Notes

7.14.19 24.2.2 (March 2014)

Changed

- null string is treated as an invalid key for all record operations of GAMSSymbol.

Fixed

- a bug when creating GAMSDatabase from source database.
- a bug when initializing a variable type in GAMSVarsiable.
- a bug in GAMSSymbol: methods getVarType and getEquType.
- exit codes in GAMSGlobals.ExitCodeMessage. Improved
- GAMSModelInstance.instantiate: make sure that the method does not solve the model, but only prepares everything required for the following GAMSModelInstance.solve.

7.14.20 24.2.1 (December 2013)

New

- tutorial: (GAMS_java_Tutorial.pdf) under <Path/To/GAMS>/docs/API. From the GAMSIDE this document can be accessed at Help -> Docs -> API.
- method in GAMSGlobals.SpecialValues: setValue, to set a GAMS special value
- method in GAMSOptions: export, to write an option into a parameter file
- methods in GAMSWorkspace: addOptions to create a GAMSOptions object from either another object or an option file
- methods in GAMSWorkspace: getAPIVersion, getAPIMajorReleaseNumber, getAPIMinorReleaseNumber, and getAPIGoldReleaseNumber, to retrieve API version number

Changed

- default value of GAMSGlobals.SpecialValues: NAN, PLUS_INF, MINUS_INF, and EPS
- deprecated GAMSOptions.PoolFree4, replaced by GAMSOptions.EIntVarUp
- deprecated the type of GAMSOptions.NoNewVarEqu, replaced by enum GAMSOptions.ENoNewVarEqu

Fixed

- a bug when running a job with an input directory IDir added into GAMSOptions object.
7.14.21 24.1.3 (July 2013)

Improve data iterator

- New classes:
  - GAMSDatabaseIterator implements java.util.Iterator
  - GAMSSymbolIterator implements java.util.Iterator

- Deprecated Interface:
  - GAMSSymbolIterable, replaced by GAMSDatabaseIterator and GAMSSymbolIterator

- Changes in GAMSDatabase:
  - GAMSDatabase implements GAMSDatabaseIterator instead of GAMSSymbolIterable
  - deprecated the implemented methods of GAMSSymbolIterable: next(), hasNext(), and remove()

- Changes in GAMSSymbol:
  - GAMSSymbol implements GAMSSymbolIterator instead of GAMSSymbolIterable
  - deprecated the implemented methods of GAMSSymbolIterable: next(), hasNext(), and remove()

- New methods in GAMSSymbolRecord:
  - moveNext: to iterate to the next record using the current data iterator criterion
  - movePrevious: to iterate to the previous record using the current data iterator criterion

Update example:

- Transport12 at [Path/To/GAMS]/apifiles/Java/transport/Transport12.java

7.14.22 24.1.1 (May 2013)

Changes in GAMSDatabase:

- deprecates the compact method, as it has no effect anymore.

Changes in GAMSGlobals:

- the default value of working directory has been changed from System.getProperty("user.dir") to System.getProperty("java.io.tmpdir").

Changes in GAMSMModelInstance:

- deprecates the instantiate(GAMSOptions options), instantiate(GAMSModifier[]), and instantiate(GAMSOptions, GAMSModifier[]) methods and replaced by instantiate(String, GAMSModifier ...) and instantiate(String, GAMSOptions, GAMSModifier ...) methods

Changes in GAMSWorkspace and GAMSWorkspaceInfo:
• deprecates boolean debug flag and replaced by a debug level flag (type of a new class \texttt{GAMSGlobal.s.DebugLevel}).

• allows an override of debug level flag from an environment variable \texttt{GAMS00APIDebug}

New methods in \texttt{GAMSDatabase}:

- \texttt{GAMSDatabase.addEquation}, \texttt{addParameter}, \texttt{addSet}, and \texttt{addVariable}: to add symbols with domain information.

- \texttt{checkDomains}: to check whether or not all records of all symbols are within the specified domain of the symbols.

- \texttt{getDatabaseDomainViolations}: to retrieve a domain violation information as a list of \texttt{GAMSDatabaseDomainViolation} objects.

- \texttt{isAutoDomainCheckingSuppressed} and \texttt{suppressAutoDomainChecking}: to control whether domain checking will be called when exporting a database.

New class \texttt{GAMSDatabaseDomainViolation}:

- contains domain violation information of all symbols (if any) in the database.

- returns call from a new method \texttt{GAMSDatabase.getDatabaseDomainViolations}.

New enumeration class \texttt{GAMSGlobal.s.DebugLevel}:

- defines values of different GAMS Debug Levels.

New methods in \texttt{GAMSModelInstance}:

- \texttt{copyModelInstance}: to copy a \texttt{GAMSModelInstance} object.

- \texttt{interrupt}: to send an interrupt signal to a running \texttt{GAMSModelInstance} object.

New enumerated value of \texttt{GAMSModelInstance.SymbolUpdateType}:

- \texttt{GAMSModelInstance.SymbolUpdateType.INHERIT}: to specify \texttt{GAMSModelInstance.SymbolUpdateType} separately for each \texttt{GAMSModifier}.

New methods in \texttt{GAMSModifier}:

- \texttt{constructor}: to specify \texttt{SymbolUpdateType} for each \texttt{GAMSModifier} object.

- \texttt{getUpdateType}: to retrieve \texttt{SymbolUpdateType} property of the object.

New methods in \texttt{GAMSSymbol}:

- \texttt{checkDomains}: to check whether or not all records of the symbol are within the specified domain.

- \texttt{getDomains}: to retrieve a list of domains of the symbol, each element is either a \texttt{GAMSSet} (real domain) or a \texttt{String} (relaxed domain).
• **getSymbolDomainViolations**: to retrieve a domain violation information as a list of `GAMSSymbolDomainViolation` objects.

• **mergeRecord**: to add a new symbol record in case the record does not exist.

**New class** `GAMSSymbolDomainViolation`:

• contains domain violation information of the symbol (if any).

• returns call from a new method `GAMSSymbol.getSymbolDomainViolations`.

**New methods in** `GAMSWorkspace`

• **getGAMSVersion**: to retrieve information about GAMS Version.

• **getGoldReleaseNumber**: to retrieve GAMS GOLD Release Number.

• **getMajorReleaseNumber**: to retrieve GAMS Major Release Number.

• **getMinorReleaseNumber**: to retrieve GAMS Minor Release Number.

Fixed a bug when iterating through the records of a `GAMSSymbol`

**Changes of location of examples**:

- **Bender Examples**: from `[Path/To/GAMS]/apifiles/Java/Benders*.java` to `[Path/To/GAMS]/apifiles/Java/benders/Benders*.java`

- **Cutstock**: from `[Path/To/GAMS]/apifiles/Java/Custock.java` to `[Path/To/GAMS]/apifiles/Java/cutstock/Cutstock.java`

- **ConsoleInterrupt**: from `[Path/To/GAMS]/apifiles/Java/ConsoleInterrupt.java` to `[Path/To/GAMS]/apifiles/Java/interrupt/ConsoleInterrupt.java`

- **Transport Examples**: from `[Path/To/GAMS]/apifiles/Java/Transport*.java` to `[Path/To/GAMS]/apifiles/Java/transport/Transport*.java`

- **Warehouse**: from `[Path/To/GAMS]/apifiles/Java/Warehouse.java` to `[Path/To/GAMS]/apifiles/Java/warehouse/Warehouse.java`

**New examples**:

- **SimpleCutstock** at `[Path/To/GAMS]/apifiles/Java/cutstock/SimpleCutstock.java`

- **DomainCheck** at `[Path/To/GAMS]/apifiles/Java/domain/DomainCheck.java`

- **Transport13** at `[Path/To/GAMS]/apifiles/Java/transport/Transport13.java`

- **Tsp** at `[Path/To/GAMS]/apifiles/Java/tsp/Tsp.java`

**7.14.23 24.0.2 (February 2013)**

**New method in** `GAMSSymbol`

• added new method `copySymbol`. 
7.14.24  24.0.1 (December 2012)

This release contains a beta version of the object-oriented Java API that can be used to control GAMS from a Java program. It allows the seamless integration of GAMS into Java by providing appropriate classes for the interaction with GAMS. GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in the more efficient way (GAMSModelInstance).

- A Java program that uses object-oriented Java API requires at least Java SE 5 to compile and run.
- All classes are distributed within one single jar file GAMSJavaAPI.jar with a namespace com.gams.api, located under the [Path/To/GAMS]/apifiles/Java/api/ directory.
- Java program examples are distributed with namespace com.gams.examples, located under [Path/To/GAMS]/apifiles/Java/ directory.
- Installation and detailed documents can be found in [Path/To/GAMS]/apifiles/readme.txt and [Path/To/GAMS]/docs/API/GAMS.java.pdf.
- Javadoc for GAMSJavaAPI.jar can be found under [Path/To/GAMS]/apifiles/java/api/javadoc directory.

7.15  Release Notes

7.15.1  40.1.0

- Dropped support for Python 3.6.

7.15.2  39.1.0

- Added GamsSet.settype property to distinguish Multi and Singleton sets.
- New optional argument settype=SetType.Multi in GamsDatabase.add_set() and GamsDatabase.add_set_dc()

7.15.3  38.1.0

- Added support for Python 3.10.

7.15.4  35.1.0

- Fixed GamsOptions.output parameter being ignored in GamsJob.run().

7.15.5  34.1.0

- Added support for Python 3.9.
7.15.6 32.1.0

- Dropped support for Python 2.
- Automatically finding a GAMS installation from the Windows registry has been changed from using
  [HKEY_CURRENT_USER\HKEY_CLASSES_ROOT]\gams.location to [HKEY_CURRENT_USER\HKEY_LOCAL_MACHINE]\Software

7.15.7 31.1.0

- The object-oriented GAMS Python API comes now in source code using the MIT open source license.
- For accessing the GAMS API the Python interpreter needs to find files in <GAMS Systemdirectory>/apifiles/Python/api
  and <GAMS Systemdirectory>/apifiles/Python/gams where XY corresponds to the Python version X.Y, e.g. 3.7. The GMSPython installation is already set up to find all required files. The documentation section Let Python find the GAMS Python files gives details on how to connect an external Python system with GAMS.
- There is a different directory structure in <GAMS Systemdirectory>/apifiles/Python. The content of the api directory has been moved to different places:
  - api.27 for the binary Python 2.7 API files
  - src for the files required to build the expert-level API files from source
  - gams for source files of the GAMS Python API plus a few extra Python source files for working with indexed EMP syntax, e.g. emppython1.gms : Three Simple EMP Models with Indexed EMP Syntax and Python Parser
  - thirdparty for the Python package ply for processing of indexed EMP syntax
- Option pySetup has been removed from the class GamsOptions.
- Removed GAMS return code 116 ("Could not find GMSPython").
- Fixed a memory leak with GamsModelInstance.solve.

7.15.8 30.1.0

- Dropped support for Python 3.4.
- Added support for Python 3.8.

7.15.9 29.1.0

- Added support for Python 3.7.

7.15.10 28.1.0

- Dropped support for Python 2.6.
7.15.11  25.0.1

- Added implementation of \_eq\_() and \_ne\_() to classes \_GamsSymbol and \_GamsSymbolRecord
  and its derived classes. As a result the behavior of the operators == and != has changed. == returns
  now True, if the internal data reference is the same. The behavior of is remains unchanged.
- New example transport\_gdx\_py that shows how to import and export GDX files.
- Added description of example transport\_gdx\_py to the tutorial.

7.15.12  24.9.2

- Fixed a bug in the setup script that prevented files from being installed in the correct location in
  certain cases.

7.15.13  24.9.1

- New functions GamsWorkspace.add\_job\_from\_apilib, GamsWorkspace.add\_job\_from\_noalib to create
  GamsJob from models from the GAMS API Library and the Nonlinear Optimization Applications
  Library
- Added a version check for the setup\_py scripts to avoid unintentional installation of wrong versions

7.15.14  24.8.4

- Added support for Python 3.6
- Fixed a bug in GamsJob.run() that prevented the underlying GAMS model from terminating, if
  executables spawned by GAMS generate log output that is not captured.

7.15.15  24.8.1

- The option GamsOptions.errorlog was renamed as GamsOptions.errorlog in order to indicate it as
  public. The option sets the maximum error message lines written to the log for each error

7.15.16  24.7.4

- Added support for Python 3.4 on Mac OS X
- Skip creation of a GDX file in GamsModelInstance.instantiate that was unreachable from within the
  API.
- Fixed a problems with Exceptions in GamsWorkspace.

7.15.17  24.7.2

- Fixed a minor bug regarding the names of GamsJob listing files.
7.15.18 24.7.1

- Fixed a bug with the property GamsOptions.defines: When too many entries were added, all of them were ignored
- Fixed a memory leak in GamsDatabase

7.15.19 24.6.1

- New Python example transport8a.py

7.15.20 24.5.2

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. The property GamsWorkspace.system_directory now always returns the canonical path with all symbolic links resolved.

7.15.21 24.5.1

- Added support for Python 3.4 (Windows and Linux only)
  - The examples have been changed to be compatible with all supported Python versions
- New Python example special_values.py.
- New Python example clad.py.
- New functions GamsWorkspace.noalib and GamsWorkspace.apilib that can be used to retrieve models from the NOALIB and the APILIB

7.15.22 24.4.2

- Fixed a bug in the constructor of all subclasses of GamsSymbol that occurred when the explanatory text was omitted.
- Disable unwanted debug output from Couenne when running with GamsModelInstance
- New property _GamsSymbol.domains_as_strings: Domains of Symbol, each element is a string. Note: If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set
- Fixed potential problem with GamsModelInstance used with BARON.

7.15.23 24.4.1

- Fixed a bug in GamsDatabase.merge_record that prevented the function from creating a record if none was found.
- We changed the handling of GAMS Aliases in the object oriented APIs:
  - If we ask for the number of GamsSymbols in a GamsDatabase, the Aliases will be excluded.
  - If we iterate over all GamsSymbols in a GamsDatabase, Aliases will be skipped.
  - If we ask explicitly for an Alias in a GamsDatabase (GamsDatabase.get_set("a") with a being an Alias) we will get a reference to the GamsSet referenced by the Alias, not the Alias itself.
  - Note: Aliases can appear in a GamsDatabase only, if it was initialized by a GDX file containing an Alias.
  - The new Python example alias.py demonstrates this new behavior.
7.15.24 24.3.2

- Make more GAMS options available through the GamsOptions class (Removed leading underscores of property identifiers):
  - GamsOptions.appendexpand: Expand file append option
  - GamsOptions.appendout: Output file append option
  - GamsOptions.dumpopt: Writes preprocessed input to the file input.dmp
  - GamsOptions.dumpparms: GAMS parameter logging
  - GamsOptions.errmsg: Placing of compilation error messages
  - GamsOptions.expand: Expanded (include) input file name
  - GamsOptions.ferr: Alternative error message file
  - GamsOptions.jobtrace: Job trace string to be written to the trace file at the end of a Gams job
  - GamsOptions.limcol: Maximum number of columns listed in one variable block
  - GamsOptions.limrow: Maximum number of rows listed in one equation block
  - GamsOptions.logline: Amount of line tracing to the log file
  - GamsOptions.on115: Generate errors for unknown unique element in an equation
  - GamsOptions.output: Output file
  - GamsOptions.pagecontr: Output file page control option
  - GamsOptions.pagesize: Output file page size (=0 no paging)
  - GamsOptions.pagewidth: Output file page width
  - GamsOptions.reference: Symbol reference file
  - GamsOptions.scriptexit: Program or script to be executed at the end of a GAMS run
  - GamsOptions.suppress: Compiler listing option
  - GamsOptions.symbol: Symbol table file
  - GamsOptions.tracelevel: Solvestat threshold used in conjunction with a=GT

7.15.25 24.3.1

- On Linux and OS X: LD_LIBRARY_PATH and DYLD_LIBRARY_PATH do not need to contain a GAMS system directory anymore. Anyway, if no system directory is specified in the GamsWorkspace constructor, PATH and LD_LIBRARY_PATH/DYLD_LIBRARY_PATH is used to find a GAMS system directory. The order is as follows:
  - Linux: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check LD_LIBRARY_PATH
  - OS X: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check DYLD_LIBRARY_PATH
  - Windows: If no system directory is specified in the GamsWorkspace constructor, check registry.
- GamsJob.run creates out.db now even if it throws a GamsExceptionExecution
- New example transport14
- New function GamsSymbolRecord.key(int index): Retrieve key of GamsSymbolRecord on position index
- GamsModelInstance.solve now also works for solvers which require a subsolver, e.g. Dicopt
7.15.26 24.2.3

- Fix memory leak in GamsDatabase

7.15.27 24.2.2

- Fixed a bug regarding the subtype of equations and their default records, when the equation was added to a GamsDatabase from the API.
- Fixed a bug in GamsSymbol.delete_record.
- Fixed GamsExitCode.
- Make sure that GamsModelInstance.instantiate() does not solve the model, but only prepares everything required for the following solve()sus

7.15.28 24.2.1

- Add tutorial (GAMS_python_Tutorial.pdf) to <Path/To/GAMS>/docs/API. From the GAMSIDE document can be accessed at Help -> Docs -> API
- New parameter opt_file in function GamsWorkspace.add_options allows to create an instance of GamsOptions that is initialized by an existing option file
- New function GamsOptions.export: Write GamsOptions into a parameter file
- New properties: GamsWorkspace.api_version, GamsWorkspace.api_major_rel_number, GamsWorkspace.api_minor_rel_number, and GamsWorkspace.api_gold_rel_number
- New property GamsWorkspace.my_eps: Reset value to be stored in and read from GamsDatabase for Epsilon.
- New class NoNewVarEqu providing static fields to set option nonewarequ
- Renaming of GamsOptions.PoolFree4 to GamsOptions.IntVarUp and property poolfree4 to intvarup
- Fixed a bug in _GamsSymbol.add_record() that occurred when using empty strings as keys
- Fixed a bug in GamsModelInstance.solve() that affected the logging behavior
- The following functions throw an exception, when a wrong data type is passed for the keys/slice parameter: delete_record, find_record, add_record, merge_record, first_record, last_record. Valid data types are str, list, tuple and their subclasses

7.15.29 24.1.1

- New examples: markowitz.py, tsp.py, transport13.py, simple_cutstock.py, domain_checking.py
- New facility for GamsSymbols with domain information:
  - GamsDatabase:
    * New functions add_parameter_dc, add_variable_dc, add_equation_dc and add_set_dc: Create GamsSymbols with domain information
    * New function check_domains: Check for all symbols if all records are within the specified domain of the symbol
* New function get_database_dvs: Return all GamsDatabaseDomainViolations
* New property suppress_auto_domain_checking: Controls whether domain checking is called in GamsDatabase export

- GamsSymbol and its derived classes:
  * New function check_domains: Check if all records are within the specified domain of the symbol
  * New function get_symbol_dvs: Return all GamsDatabaseDomainViolations
  * New property domains: Domains of Symbol, each element is either a GamsSet (real domain) or a string (relaxed domain)

- New classes GamsDatabaseDomainViolation and GamsSymbolDomainViolation that are returned by GamsDatabase.get_database_dvs and GamsSymbol.get_symbol_dvs

- New functions in GamsModelInstance:
  - copy_modelinstance: Copies a GamsModelInstance to a new GamsModelInstance which gets constructed at this call
  - interrupt: Sends interrupt signal to running GamsModelInstance

- New function in GamsSymbol and its derived classes: merge_record finds a record if it exists and adds it if not

- New functions in GamsWorkspace: add_job_from_gamslib, add_job_from_testlib, add_job_from_emplib, add_job_from_datalib and add_job_from_finlib

- New properties in GamsWorkspace: version, major_rel_number, minor_rel_number, gold_rel_number

- New optional parameter in model_name for functions add_database_from.gdx/add_database in GamsWorkspace: GAMS string constant that is used to access this database

- Changed the debug argument passed to the GamsWorkspace constructor. Use members of class DebugLevel: Off, KeepFiles, ShowLog and Verbose instead of True and False

- New sub class of GamsException: GamsExceptionExecution, which provides additional info about the reason of the failed execution

- Allow to define SymbolUpdateType for each GamsModifier seperately

- Significant performance improvement for function GamsWorkspace.add_database when creating from an already existing database

- Changed the unicode settings on Linux from UCS2 to UCS4

- The Debug parameter of GamsWorkspace can be overwritten by the system environment variable "GAMSOOAPIDEBUG" if set to one of the following: Off, KeepFiles, ShowLog, Verbose

- The compact method of GamsDatabase is obsolete and has no effect anymore. It will be removed in the future.

- Fixed a bug in GamsModelInstance.solve()

- Fixed a bug in GamsDatabase, where the special value for undefined was set to 0 instead of 1.0E300 (SV_UNDEF)

- Fixed a bug when iterating through the records of a GamsSymbol

7.15.30  24.0.2

- Added new function _GamsSymbol.copy_symbol which copys all records from one to another GamsSymbol

- Label and symbol lookup is now case insensitive

- Fixed handling of infinite bounds for GamsModifiers
This release contains a beta version of the object-oriented Python API that can be used to control GAMS from within Python 2.7. It allows the seamless integration of GAMS into Python by providing appropriate classes for the interaction with GAMS. The GamsDatabase class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the GamsJob class and by using the GamsModelInstance class a sequence of closely related model instances can be solved in an efficient way. To use this API see the Getting started section.

### 7.16 GAMS Environment Object Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgFileType</td>
<td>File type of scratch files</td>
<td>1</td>
</tr>
<tr>
<td>Cheat</td>
<td>optCheat solver default</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([-\infty, \infty])</td>
<td></td>
</tr>
<tr>
<td>CurSolver</td>
<td>Solver id of the last gecallsolver</td>
<td>0</td>
</tr>
<tr>
<td>CutOff</td>
<td>CutOff solver default</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([-\infty, \infty])</td>
<td></td>
</tr>
<tr>
<td>DomLim</td>
<td>Domain violation limit solver default</td>
<td>0</td>
</tr>
<tr>
<td>FDDelta</td>
<td>Step size for finite differences</td>
<td>1e-5</td>
</tr>
<tr>
<td></td>
<td>Range: ([1e-9, 1])</td>
<td></td>
</tr>
<tr>
<td>FDOpt</td>
<td>Options for finite differences</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 14])</td>
<td></td>
</tr>
<tr>
<td>GamsVersion</td>
<td>GAMS CMEX Version</td>
<td>999</td>
</tr>
<tr>
<td>GenSolver</td>
<td>Indicator for generic solver executable</td>
<td>0</td>
</tr>
<tr>
<td>HeapLimit</td>
<td>Maximum Heap size allowed</td>
<td>1e20</td>
</tr>
<tr>
<td>IDEFFlag</td>
<td>IDE Flag</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 unknown environment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 runs under GAMS IDE</td>
<td></td>
</tr>
<tr>
<td>Integer1</td>
<td>Range: ([-\infty, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>Integer2</td>
<td>Range: ([-\infty, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>Integer3</td>
<td>Range: ([-\infty, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>Integer4</td>
<td>Range: ([-\infty, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>Integer5</td>
<td>Range: ([-\infty, \infty])</td>
<td>0</td>
</tr>
<tr>
<td>IterLim</td>
<td>Iteration limit solver default</td>
<td>2000000000</td>
</tr>
<tr>
<td>Keep</td>
<td>Keep scratch directory</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([0, 1])</td>
<td></td>
</tr>
<tr>
<td>License1</td>
<td>License line 1</td>
<td></td>
</tr>
<tr>
<td>License2</td>
<td>License line 2</td>
<td></td>
</tr>
<tr>
<td>License3</td>
<td>License line 3</td>
<td></td>
</tr>
<tr>
<td>License4</td>
<td>License line 4</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>License5</td>
<td>License line 5</td>
<td></td>
</tr>
<tr>
<td>LogOption</td>
<td>Log option</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 no log output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 log output to screen (console)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 log output to logfile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 log output to standard output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 log output to logfile and standard output</td>
<td></td>
</tr>
<tr>
<td>NameCtrFile</td>
<td>Control file name</td>
<td></td>
</tr>
<tr>
<td>NameCurDir</td>
<td>Current directory</td>
<td></td>
</tr>
<tr>
<td>NameExtFFile</td>
<td>External Function file name</td>
<td></td>
</tr>
<tr>
<td>NameGamsDate</td>
<td>GAMS Date</td>
<td></td>
</tr>
<tr>
<td>NameGamsTime</td>
<td>GAMS Time</td>
<td></td>
</tr>
<tr>
<td>NameInstr</td>
<td>Instruction file name</td>
<td></td>
</tr>
<tr>
<td>NameLogFile</td>
<td>Log file name</td>
<td></td>
</tr>
<tr>
<td>NameMatrix</td>
<td>Matrix file name</td>
<td></td>
</tr>
<tr>
<td>NameParams</td>
<td>Parameter file name</td>
<td></td>
</tr>
<tr>
<td>NameScenFile</td>
<td>Scenario file name</td>
<td></td>
</tr>
<tr>
<td>NameScrDir</td>
<td>Scratch directory</td>
<td></td>
</tr>
<tr>
<td>NameScrExt</td>
<td>Scratch extension</td>
<td>dat</td>
</tr>
<tr>
<td>NameStaFile</td>
<td>Status file name</td>
<td></td>
</tr>
<tr>
<td>NameSysDir</td>
<td>System directory</td>
<td></td>
</tr>
<tr>
<td>NameWrkDir</td>
<td>Working directory</td>
<td></td>
</tr>
<tr>
<td>NodeLim</td>
<td>Node limit solver default</td>
<td>0</td>
</tr>
<tr>
<td>OptCA</td>
<td>optAbsolute Optimality criterion solver default</td>
<td>0</td>
</tr>
<tr>
<td>OptCR</td>
<td>optRelative Optimality criterion solver default</td>
<td>0.1</td>
</tr>
<tr>
<td>PageSize</td>
<td>Output file page size</td>
<td>78</td>
</tr>
<tr>
<td>PageWidth</td>
<td>Output file page width</td>
<td>123</td>
</tr>
<tr>
<td>Real1</td>
<td>Range: $[-\infty, \infty]$</td>
<td>0</td>
</tr>
<tr>
<td>Real2</td>
<td>Range: $[-\infty, \infty]$</td>
<td>0</td>
</tr>
<tr>
<td>Real3</td>
<td>Range: $[-\infty, \infty]$</td>
<td>0</td>
</tr>
<tr>
<td>Real4</td>
<td>Range: $[-\infty, \infty]$</td>
<td>0</td>
</tr>
<tr>
<td>Real5</td>
<td>Range: $[-\infty, \infty]$</td>
<td>0</td>
</tr>
<tr>
<td>Reform</td>
<td>Reform option</td>
<td>100</td>
</tr>
<tr>
<td>ResLim</td>
<td>Resource (CPU) solver default limit</td>
<td>1000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SysOut</td>
<td>Solver Status file reporting option</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>ThreadsRaw</td>
<td>Range: [-\infty, \infty]</td>
<td>1</td>
</tr>
<tr>
<td>TryInt</td>
<td>TryInt solver default</td>
<td>0</td>
</tr>
<tr>
<td>UseCheat</td>
<td>UseCheat solver default</td>
<td>0</td>
</tr>
<tr>
<td>UseCutOff</td>
<td>UseCutOff solver default</td>
<td>0</td>
</tr>
<tr>
<td>WorkFactor</td>
<td>Work space multiplier for some solvers</td>
<td>1</td>
</tr>
<tr>
<td>WorkSpace</td>
<td>Work space for some solvers</td>
<td>0</td>
</tr>
</tbody>
</table>

7.17 GAMS Modeling Object Design

This document gives a brief introduction to GAMS' new model-solver API, including a description of its design goals and philosophy. This document is not intended to be used as a user's manual but rather to give a picture of what the new API can do and a description of some of the conceptual underpinnings for the new API.

Further material: Presentation at ICS 2011

7.17.1 Introduction

Historically, there have been a number of different GAMS solver interface libraries, i.e. libraries that enabled a solver to get all required information from a GAMS model, compute the solution, and return the solution, status information, and other information (e.g. solver log messages, nonlinear function evaluation error messages, node counts, time used) to GAMS. Each library was unique in some way, e.g. in the computer language or subset of GAMS model types it supported. Each library had its own advantages and disadvantages, but all of them were feature-poor and supported a limited scheme of operation: GAMS/Base would write scratch files containing the model to be solved, the GAMS solver link executable would use one of the solver interface libraries to read the model from scratch file and write a solution file, and GAMS/Base would read this solution file. While this scheme worked for many years it was inconvenient for the library user, difficult to maintain, slow, and a barrier to further development.

To address these issues and others, GAMS developed a new model-solver interface library, called GMO (GAMS Model Object), and a companion library called GEV (GAMS Environment Object). The two libraries are distinct and help to separate the purely model-specific tasks and information handled by GMO from the computing and solver environment that the model will be solved in, handled by GEV. The two libraries are typically used together and we will sometimes refer to them as simply GMO. For legacy reasons the new libraries support the old file-based scheme to communicate a model through scratch files.

In what follows, we first describe the basic structure, organization, and usage of GMO (Basic organization). Client applications can be split into two types: those defining or updating the model (Modeler Access to GMO) and those passing the model to a solver (Solver Access to GMO). We next describe some ways to update a model in Updating a GMO instance.
7.17 GAMS Modeling Object Design

7.17.2 Basic organization

In this section we outline the basic organization of GMO. GMO is implemented as a shared library that interfaces to different languages via language-specific header files and interface code. The header files and code are included in source form with the GAMS distribution. Since the interface code employs dynamic loading, each GMO session begins with a call to initialize the required shared library (i.e. load it and all its symbols) and return an empty model object.

The next step is to load a model into the model object. This is typically done with a single wrapper call, as in the case of loading a model from model scratch files prior to solving in one of the GAMS solver links. It is also possible build up a model from scratch with a sequence of calls to provide the model column-wise or row-wise. If the model is built up column-wise or row-wise, additional information (e.g. about nonlinearities) may be passed to the object after the basic structure is built up. Once the model is complete, a finalization routine is called. This routine prepares the model object for efficient access by the user. Typically the model does not change after this point. There are exceptions to this (see Updating a GMO instance), but they usually involve small modifications to the existing model (like adding alternative bounds) or the addition of constraints and variables that require the re-finalization of the model.

When a model is fully loaded, the GMO client can start accessing the model. Typically the GMO client doing this will be a solver, but other clients (e.g. the CONVERT solver, the Examiner utility, or a model structure browser) exist also. In addition to querying the model, the client can also request that the model be presented or viewed in various different ways. For example, GMO can be requested to use an objective function or an objective variable. Most types of information are available immediately when the model is loaded, but some type of queries (e.g. Hessians, alternative QP forms) are less common and relatively expensive in time and/or space: these require that an initialization routine be called before the information is requested. Once the client has completed its work it reports on the solution found, if any. Typically, the model object is destroyed at this point. Model access by a client, especially solvers, is an important topic that is discussed in Solver Access to GMO.

The variety of data that go into the interface library when a model is defined and that a solver can request during solution is large, but the data can be divided into two types - data defining the model (e.g. number of rows and columns, coefficients, nonlinear functions) and data defining the solver environment (e.g. available and default GAMS solvers, license info, algorithm tolerances and limits). It is sometimes useful to separate these two types of data, so that it is possible to define and work with a model without specifying anything about the environment it exists in. The environment can be specified separately, and multiple models can make use of the same basic environment. To facilitate this, the solver interface is in two parts: the GMO interface deals with model-specific information and the GEV interface with information about the environment. The two environments are linked, since they are typically used together, with one (or multiple) models existing in one environment, but the separation makes it possible to define a model without defining its environment, etc. Since the bulk of the information and the code lies with GMO, we usually speak of the two libraries as one, called GMO, but it is worth remembering that GEV is a separate object.

7.17.3 Solver Access to GMO

One of the primary purposes of GMO is to provide convenient, efficient access to the model. GAMS links to many solvers, each solver link using GMO to access the model in different ways, so there is a wide array of possible methods to access the model in GMO. We describe them in this section.

There are multiple but equivalent views possible for the same model. For example, the model can be formulated with an objective function or with an objective variable. Free rows can be removed or left in, and the rows and/or columns can be permuted. Prior to accessing the model, the solver link can choose what view it would like to use. The view can be changed later, but since this can obviate the data obtained previously, the choice of view is usually made early on and left unchanged. Moreover, a view can be stored and restored in case multiple users access the same model.
Application Programming Interfaces

All solvers will need to get basic information about a model - simple things like row and column counts, nonzero counts, etc. These are available via simple access routines. Since the model is essentially fixed (assuming the view does not change), basic information is pre-computed when the model takes its final form and is available at essentially no cost. Information about the rows and columns (e.g. variable level values, marginal values, variable priorities, row types) is also available, either for single rows/cols or in an array for all rows/cols at once.

For linear models, the only information left to get is the matrix of coefficients. This is available in a number of different forms e.g. column-wise or row-wise, either as a complete matrix returned with one call or as separate rows or columns returned via many calls. For nonlinear models, there are more variants on this theme: calls to return the function value or the function value and derivatives, calls for constraints and calls for the objective function. You can request that the nonlinear model be treated as its linearization around a certain point. There are also routines to do interval evaluations of functions and gradients. The GMO library prepares the model object to handle all these calls efficiently when the model takes its final form.

Nonlinear models present several additional challenges. When NL functions are evaluated, mathematical errors can occur (e.g. sqrt(-1)). GMO supports different ways to log and handle these errors. Some algorithms will require or want 2nd-order information. Since this can be relatively expensive to compute, the solver must first call an initialization routine specifying what is desired (e.g. only Hessian-vector products, Lagrangian Hessian, single-row Hessians) and potentially a limit on the memory dedicated to computing Hessians. After this initialization, 2nd-order information is available.

Once the solver finishes, the solution (if any) and related status values must be reported back to GMO. There are a number of convenience routines included in GMO for this purpose.

A convenient interface was and is an important design goal for GMO, especially in the area of solver access to the model. Convenience takes many forms: a number of ways to get the same model data in order to support a variety of different solver libraries, self-contained calls with minimal requirements on previous setup for correct behavior, and a modern interface adhering to design principles like information hiding, encapsulation, and an object model. By keeping these goals at the fore when designing GMO we have made accessing the model convenient and efficient, which greatly simplifies the task of building a correctly-functioning solver link.

In addition to providing convenient access to the model, a solver link needs to expose a particular API to be recognized by the GAMS system. This is not directly connected to the GMO API itself but knowledge about the expected solver link library API is required so we include it here.

GAMS calls a solver through the GEV API function `gevCallSolver` which eventually results in a sequence of calls to functions in the solver link library. GAMS loads solver link libraries dynamically at run time. The name of the library is specified in the configuration file together with the three letter audit code, e.g., `cpx` for CPLEX. All functions in the solver link library are prefixed by this audit code (we use `xyz` in this example). The following functions are recognized:

- **void xyzInitialize(void)** (optional): Function called when loading the solver link library.
- **void xyzFinalize(void)** (optional): Function called when unloading the solver link library.
- **int xyzCreate(void**∗∗**handle, char**∗**msg, int sizeBuf)**: Function to create a solver link object. Returns 0 and stores a non-NULL pointer to the object in `handle` on success, otherwise an error message should be stored in `msg`.
- **void xyzFree(void**∗∗**handle)**: Function to destroy a solver link object.

Depending on the solver interface type (again communicated as part of the configuration file) different functions need to be in the API. For solver interface type 1 we have:

- **int xyzReadyAPI(void**+**handle, void**+**gmoHandle)**: Function to set up the model instance in the solver space. Returns 0 on success.
• int xyzCallSolver(void* handle): Function to do the actual solve and solution reporting. Returns 0 on success.

• int xyzModifyProblem(void* handle) (optional): Function to modify the problem. Returns 0 on success.

For solver interface type 2 we only have a single call:

• int xyzCallSolver(void* handle, void* gmoHandle): Function to setup and solve. Returns 0 on success.

Usually, a solve in GAMS triggers the following sequence of function: xyzCreate, xyzReadyAPI, xyzCallSolver, xyzFree (interface type 1) or xyzCreate, xyzCallSolver, xyzFree (interface type 2) and the separation of xyzReadyAPI and xyzCallSolver seems unnecessary.

With Gather-Update-Solve-Scatter (GUSS) and the GAMSModelInstance class in various APIs we repeatedly solve a model with the same rim but different data (bounds, rhs, matrix coefficients). In this case the sequence of solves results in a sequence of function calls. Without the presence of xyzModifyProblem the following sequences will be called: xyzCreate, xyzReadyAPI, xyzCallSolver, xyzReadyAPI, xyzCallSolver, ..., xyzFree (interface type 1) or xyzCreate, xyzCallSolver, xyzCallSolver, xyzCallSolver, ..., xyzFree (interface type 2). So, again there is little point (but possible) to split the link work in xyzReadyAPI and xyzCallSolver without the presence of xyzModifyProblem. If xyzModifyProblem is present (only with flavor 1) the following sequences will be called: xyzCreate, xyzReadyAPI, xyzCallSolver, xyzModifyProblem, xyzCallSolver, xyzModifyProblem, xyzCallSolver, ..., xyzFree.

7.17.4 Modeler Access to GMO

In normal operation the modeler creates a model in GAMS source form and the GAMS/Base module handles all the details necessary to build up model instances in GMO. However, there are cases (e.g. solving a model many times over with different scenario data) where it is useful to access and modify a GMO instance outside of GAMS/Base. GMO provides routines to support this, specifically routines to update the model (see Updating a GMO instance), solve the model with one of the GAMS solvers available as part of the GAMS environment and GEV, and retrieve the solution.

Access to a GMO instance is usually done using integer indices, [1..m] for the rows and [1..n] for the columns. However, it is also possible to reference rows and columns using the notation from the original GAMS model, e.g. x('seattle', 'topeka') or demand('chicago'). This is more convenient when specifying what data should be updated to define a new scenario. In order to update a model instance in this case, it is useful to have a convenient, efficient way to translate row and column indices from one form to the other. This translation is done by the model dictionary interface DCT.

The DCT API provides information about the variables, equations, and sets used to define the GMO model instance. In addition, given a row or column index, it will translate this index into the equation or variable name and the list of set labels that correspond to this index. It will also do the inverse operation, translating an equation/variable name and list of labels into a row or column index. GAMS builds models in a way that makes this translation very efficient.

This organization is also useful when updating models to create new scenarios. Models are organized so that:

• There is one ordered universe of all labels appearing in the variables and equations of a given model. The order for the labeling does not change from symbol to symbol within a model.
• All of the columns in a model corresponding to a given variable are contiguous. (Ditto for rows and equations)

• The columns corresponding to a given variable are sorted by their index labels, using the universal order, with the left-most index varying slowest. (Ditto for rows and equations)

• GDX data is sorted in the same way: sorted by index label order, left-most index varying slowest. This can be very useful, provided the model and the GDX data use the same label ordering.

Using the various GAMS component libraries, it is possible to create a model updating interface that is both convenient for the user (by using the symbolic names referenced in the GAMS source) and efficient in updating the model data.

7.17.5 Updating a GMO instance

There are several possible ways to modify a model contained in a GMO instance. By updating, we mean an actual model change, rather than a change in the view that doesn't alter the set of solutions.

In perhaps the simplest case, the user can specify alternate variable bounds, as is useful when implementing a branch and bound code. The original bounds are retained in this case, so that the model can be reverted to the original bounds with a single call.

It is possible to linearize a model around a given point. This requires no information from the point to linearize around and can be undone easily. This is almost a change in view but since the solutions of the original and linearized models may be quite different we can also consider this a model update.

Many algorithms (e.g. outer approximation, Benders decomposition, Danzig-Wolfe decomposition) involve the addition of rows, and columns incident on these rows, to a core model. Such algorithms can be implemented effectively using GMO by taking advantage of GMO facilities to add rows and columns. There will be only a small amount of work required to prepare the GMO instance for use after each round of additional rows and columns. Note that all the changes to the model in this case are additions: the existing structure and data are left as is.

Finally, there are facilities in GMO to update variable bounds (as mentioned before), variable types (e.g. to relax single discrete variables), and the right-hand-side of constraints. The user cannot directly modify matrix coefficients. There is a much better way to safely update exogenous data in a model: GAMS can provide a model instance in which GAMS parameters are visible and can be updated. A reevaluation of the expressions in the models provides the updated matrix elements. The underlying concept is also used in Gather-Update-Solve-Scatter (GUSS) and described in more detail in this paper. This is a powerful tool to efficiently do e.g. Monte-Carlo simulation without the need for GAMS to regenerate the entire model over and over.
Index

**, reserved non-alphanumeric symbol, 535
*GCK file
  GAMSCHK, 1723
*GCK file, comments
  GAMSCHK, 1726
*GCK file, page width
  GAMSCHK, 1726
++, circular operator, 571
++, reserved non-alphanumeric symbol, 535
-, circular operator, 571
-, reserved non-alphanumeric symbol, 535
->, reserved non-alphanumeric symbol, 535
.., reserved non-alphanumeric symbol, 535
.ap
  suffixes, put-file, 996
.bm
  suffixes, put-file, 996
.case
  suffixes, put-file, 996
.cc
  suffixes, put-file, 996
.cr
  suffixes, put-file, 996
.errors
  suffixes, put-file, 996
.first
  set attributes, set, 553
.fx
  suffixes, variable, 616
.grad
  suffixes, function, 597
.gradH
  suffixes, function, 597
.gradL
  suffixes, function, 597
.gradn
  suffixes, function, 597
.hdcc
  suffixes, put-file, 996
.hdcr
  suffixes, put-file, 997
.hdll
  suffixes, put-file, 997
.hess
  suffixes, function, 597
.hessn
  suffixes, function, 597
.high
  suffixes, function, 597
.infeas
  suffixes, equation, 632
  suffixes, variable, 618
.l
  suffixes, equation, 631
  suffixes, variable, 617
.last
  set attributes, set, 553
.case
  suffixes, put-file, 997
.len
  set attributes, set, 553
.lj
  suffixes, put-file, 997
.ll
  suffixes, put-file, 997
.lm
  suffixes, put-file, 997
.lo
  suffixes, equation, 631
  suffixes, variable, 616
.low
  suffixes, function, 597
.lp
  suffixes, put-file, 997
.lw
  suffixes, put-file, 997
.m
  suffixes, equation, 631
  suffixes, variable, 617
.nd
  suffixes, put-file, 998
.nj
  suffixes, put-file, 998
.nr
  suffixes, put-file, 998
.nw
  suffixes, put-file, 998
.nz
  suffixes, put-file, 998
.off
  set attributes, set, 553
.ord
  set attributes, set, 553
.pc
  suffixes, put-file, 999
.pdir
  suffixes, put-file, 998
.pos
set attributes, set, 553
.prior
suffixes, variable, 617
.ps
suffixes, put-file, 1000
.pw
suffixes, put-file, 1000
.range
suffixes, equation, 632
suffixes, variable, 618
.rev
set attributes, set, 553
.scale
suffixes, equation, 631
suffixes, variable, 617
.sj
suffixes, put-file, 1000
.slack
suffixes, equation, 632
suffixes, variable, 618
.slacklo
suffixes, equation, 632
suffixes, variable, 618
.slackup
suffixes, equation, 632
suffixes, variable, 618
.stage
suffixes, equation, 631
suffixes, variable, 617
.sw
suffixes, put-file, 1000
.te
suffixes, identifier (put context), 1010
suffixes, string type, 591
.tf
suffixes, put-file, 1001
.tj
suffixes, put-file, 1001
.tl
suffixes, identifier (put context), 1009
suffixes, string type, 591
.tlcc
suffixes, put-file, 1001
.tlcr
suffixes, put-file, 1001
.tll
suffixes, put-file, 1001
.tm
suffixes, put-file, 1002
.tn
suffixes, identifier (put context), 1010
.ts
suffixes, identifier (put context), 1009
suffixes, string type, 591
.tw
suffixes, put-file, 1002
.uel
set attributes, set, 553
.up
suffixes, equation, 631
suffixes, variable, 616
.val
set attributes, set, 553
.value
suffixes, function, 597
.ws
suffixes, put-file, 1002
/, cursor control, 1005
=b=, reserved non-alphanumeric symbol, 535
=c=, reserved non-alphanumeric symbol, 535
=e=, reserved non-alphanumeric symbol, 535
=g=, reserved non-alphanumeric symbol, 535
=l=, reserved non-alphanumeric symbol, 535
=n=, reserved non-alphanumeric symbol, 535
=x=, reserved non-alphanumeric symbol, 535
@n, cursor control, 1005
#n, cursor control, 1005
$GDXIN, example, 445
$LOAD, example, 445
$abort, dollar control option, 878
$abort.noError, dollar control option, 878
$batInclude, dollar control option, 879
$call, dollar control option, 880
$call.Async, dollar control option, 881
$call.AsyncNC, dollar control option, 881
$call.checkErrorLevel, dollar control option, 882
$clear, dollar control option, 882
$clearError, dollar control option, 883
$clearErrors, dollar control option, 883
$comment, dollar control option, 883
$compress, dollar control option, 884
$decompress, dollar control option, 884
$dollar, dollar control option, 885
$double, dollar control option, 885
$drop, dollar control option, 886
$dropEnv, dollar control option, 886
$dropGlobal, dollar control option, 886
$dropLocal, dollar control option, 887
$echo, dollar control option, 887
$echoN, dollar control option, 887
$in, dollar control option, 888
$else, dollar control option, 888
$elseif, dollar control option, 889
$elseifE, dollar control option, 889
$elseifI, dollar control option, 889
$encrypt, dollar control option, 889
$endif, dollar control option, 890
$endCol, dollar control option, 890
$echo, dollar control option, 890
$error, dollar control option, 891
$escape, dollar control option, 891
$eval, dollar control option, 893
$eval.Set, dollar control option, 894
$evalGlobal, dollar control option, 895
$evalGlobal.Set, dollar control option, 895
$evalLocal, dollar control option, 895
$evalLocal.Set, dollar control option, 895
$\text{exit}$, dollar control option, 896
$\text{expose}$, dollar control option, 896
$\text{funcLibIn}$, dollar control option, 896
$\text{gdIn}$, dollar control option, 896
$\text{gdLoad}$, dollar control option, 897
$\text{gdOut}$, dollar control option, 897
$\text{gdUnload}$, dollar control option, 897
$\text{goto}$, dollar control option, 898
$\text{hidden}$, dollar control option, 898
$\text{hiddenCall}$, dollar control option, 899
$\text{hiddenCall.Async}$, dollar control option, 899
$\text{hiddenCall.checkErrorLevel}$, dollar control option, 899
$\text{hide}$, dollar control option, 899
$\text{if}$, dollar control option, 900
$\text{ife}$, dollar control option, 900
$\text{ifThen}$, dollar control option, 900
$\text{ifThenE}$, dollar control option, 901
$\text{ifThenI}$, dollar control option, 902
$\text{ifi}$, dollar control option, 900
$\text{include}$, 436
$\text{include}$, dollar control option, 902
$\text{inlineCom}$, dollar control option, 903
$\text{kill}$, dollar control option, 903
$\text{label}$, dollar control option, 904
$\text{libinclude}$, dollar control option, 904
$\text{lines}$, dollar control option, 905
$\text{load}$, dollar control option, 905
$\text{loadDC}$, dollar control option, 907
$\text{loadDCM}$, dollar control option, 908
$\text{loadDCR}$, dollar control option, 908
$\text{loadFiltered}$, dollar control option, 909
$\text{loadFilteredM}$, dollar control option, 909
$\text{loadFilteredR}$, dollar control option, 910
$\text{loadM}$, dollar control option, 911
$\text{loadR}$, dollar control option, 911
$\text{log}$, 439
$\text{log}$, dollar control option, 912
$\text{macro}$, dollar control option, 912
$\text{maxCol}$, dollar control option, 913
$\text{maxGoto}$, dollar control option, 913
$\text{minCol}$, dollar control option, 913
$\text{offCheckErrorLevel}$, dollar control option, 914
$\text{offDelim}$, dollar control option, 914
$\text{offDigit}$, dollar control option, 915
$\text{offDollar}$, dollar control option, 916
$\text{offDotL}$, dollar control option, 916
$\text{offDotScale}$, dollar control option, 917
$\text{offECImplicitLoad}$, dollar control option, 919
$\text{offEcho}$, dollar control option, 917
$\text{offEchoS}$, dollar control option, 917
$\text{offEchoV}$, dollar control option, 917
$\text{offEmbedded}$, dollar control option, 919
$\text{offEmbeddedCode}$, dollar control option, 920
$\text{offEmpty}$, dollar control option, 922
$\text{offEnd}$, dollar control option, 923
$\text{offEolCom}$, dollar control option, 924
$\text{offEps}$, dollar control option, 924
$\text{offEpsToZero}$, dollar control option, 926
$\text{offExpand}$, dollar control option, 927
$\text{offExternalInput}$, dollar control option, 927
$\text{offExternalOutput}$, dollar control option, 929
$\text{offFiltered}$, dollar control option, 930
$\text{offGlobal}$, dollar control option, 930
$\text{offIDCProtect}$, dollar control option, 931
$\text{offInclude}$, dollar control option, 931
$\text{offInline}$, dollar control option, 932
$\text{offListing}$, dollar control option, 932
$\text{offLocal}$, dollar control option, 933
$\text{offLog}$, dollar control option, 934
$\text{offMacro}$, dollar control option, 935
$\text{offMargin}$, dollar control option, 935
$\text{offMulti}$, dollar control option, 936
$\text{offNestCom}$, dollar control option, 939
$\text{offOEIXRef}$, dollar control option, 946
$\text{offOrder}$, dollar control option, 939
$\text{offPut}$, dollar control option, 940
$\text{offPutS}$, dollar control option, 940
$\text{offPutV}$, dollar control option, 940
$\text{offRecuse}$, dollar control option, 942
$\text{offStrictSingleton}$, dollar control option, 943
$\text{offSuffixDLVars}$, dollar control option, 943
$\text{offSymList}$, dollar control option, 944
$\text{offSymXRef}$, dollar control option, 945
$\text{offText}$, dollar control option, 945
$\text{offUEIList}$, dollar control option, 946
$\text{offUNDF}$, dollar control option, 947
$\text{offUni}$, dollar control option, 947
$\text{offUpper}$, dollar control option, 941
$\text{offVerbatim}$, dollar control option, 948
$\text{offWarning}$, dollar control option, 949
$\text{onCheckErrorLevel}$, dollar control option, 914
$\text{onDelim}$, dollar control option, 914
$\text{onDigit}$, dollar control option, 915
$\text{onDollar}$, dollar control option, 916
$\text{onDotL}$, dollar control option, 916
$\text{onDotScale}$, dollar control option, 917
$\text{onECImplicitLoad}$, dollar control option, 919
$\text{onEcho}$, dollar control option, 917
$\text{onEchoS}$, dollar control option, 917
$\text{onEchoV}$, dollar control option, 917
$\text{onEmbedded}$, dollar control option, 919
$\text{onEmbeddedCode}$, dollar control option, 920
$\text{onEmpty}$, dollar control option, 922
$\text{onEnd}$, dollar control option, 923
$\text{onEolCom}$, dollar control option, 924
$\text{onEps}$, dollar control option, 924
$\text{onEpsToZero}$, dollar control option, 926
$\text{onExpand}$, dollar control option, 927
$\text{onExternalInput}$, dollar control option, 927
$\text{onExternalOutput}$, dollar control option, 929
$\text{onFiltered}$, dollar control option, 930
$\text{onGlobal}$, dollar control option, 930
$\text{onIDCProtect}$, dollar control option, 931
$\text{onInclude}$, dollar control option, 931
$\texttt{onInline}$, dollar control option, 932
$\texttt{onListing}$, dollar control option, 932
$\texttt{onLocal}$, dollar control option, 933
$\texttt{onLog}$, dollar control option, 934
$\texttt{onMacro}$, dollar control option, 935
$\texttt{onMargin}$, dollar control option, 935
$\texttt{onMulti}$, dollar control option, 936
$\texttt{onMultiR}$, dollar control option, 937
$\texttt{onNestCom}$, dollar control option, 939
$\texttt{onOEIXRef}$, dollar control option, 946
$\texttt{onOrder}$, dollar control option, 939
$\texttt{onPut}$, dollar control option, 940
$\texttt{onPutS}$, dollar control option, 940
$\texttt{onPutV}$, dollar control option, 940
$\texttt{onRecurs}$, dollar control option, 942
$\texttt{onStrictSingleton}$, dollar control option, 943
$\texttt{onSymList}$, dollar control option, 944
$\texttt{onUEIList}$, dollar control option, 945
$\texttt{onText}$, dollar control option, 945
$\texttt{onUnD}$, dollar control option, 947
$\texttt{onUn}$, dollar control option, 947
$\texttt{onUpper}$, dollar control option, 941
$\texttt{onVerbatim}$, dollar control option, 948
$\texttt{onWarning}$, dollar control option, 949
$\texttt{phantom}$, dollar control option, 950
$\texttt{prefixPath}$, dollar control option, 951
$\texttt{protect}$, dollar control option, 952
$\texttt{purge}$, dollar control option, 952
$\texttt{remark}$, dollar control option, 952
$\texttt{sTitle}$, dollar control option, 963
$\texttt{scratchFileName}$, dollar control option, 953
$\texttt{set}$, dollar control option, 953
$\texttt{setArgs}$, dollar control option, 954
$\texttt{setComps}$, dollar control option, 955
$\texttt{setDDList}$, dollar control option, 955
$\texttt{setEnv}$, dollar control option, 956
$\texttt{setGlobal}$, dollar control option, 957
$\texttt{setLocal}$, dollar control option, 957
$\texttt{setNames}$, dollar control option, 957
$\texttt{shift}$, dollar control option, 959
$\texttt{show}$, dollar control option, 959
$\texttt{showFiles}$, dollar control option, 961
$\texttt{showMacros}$, dollar control option, 961
$\texttt{showVariables}$, dollar control option, 961
$\texttt{single}$, dollar control option, 961
$\texttt{splitOption}$, dollar control option, 962
$\texttt{stars}$, dollar control option, 963
$\texttt{stop}$, dollar control option, 963
$\texttt{sysInclude}$, dollar control option, 964
$\texttt{terminate}$, dollar control option, 964
$\texttt{title}$, dollar control option, 964
$\texttt{unLoad}$, dollar control option, 965
$\texttt{use205}$, dollar control option, 967
$\texttt{use225}$, dollar control option, 968
$\texttt{use999}$, dollar control option, 968
$\texttt{version}$, dollar control option, 968
$\texttt{warning}$, dollar control option, 969
$\texttt{\textasciitilde}$, reserved non-alphanumeric symbol, 535
abort, keyword, 532
abs, function, 586
ACCESS, data exchange
MDB2GMS, 2906
acronym, 527
acronym, keyword, 532
acronyms, keyword, 532
activity level (.l)
in equation listing, 755
use, 619
variable attribute, 616
aggregation, 704
ALAN, example from GAMSlib, 746
algorithm
Implementation of non-standard, 654
alias, 527
statement, 544
alias, keyword, 532
all, defining a model, 756
all, keyword, 532
alpha-BB
ANTIGONE, 1412
AlphaECP
convex, 1387, 1391, 1401
Cutting Plane, 1387, 1391
ECP, 1387
Extended Cutting Plane, 1387, 1390, 1391
linearization, 1390, 1391
pseudo-convex, 1387, 1391, 1401
Solver manual, 1387
AlphaECP, solver, 1387
ALUM, example from GAMSlib, 546
and, keyword, 532
and, relational operator, 658
ANDEAN, example from GAMSlib, 583, 667
ANTIGONE
alpha-BB, 1412
branch and bound global optimization, 1413
concavity, 1412
convex relaxation, 1402
convexity, 1412
edge-concavity, 1412
dependency graph, 1412
expression tree reformulation, 1412
logfile, 1403
RLT, 1412
ANTIGONE, solver, 1402
ap
suffixes, put-file, 996
API
expert-level APIs, 3041
Object-oriented APIs, 3035
Examples, 3036
Reference Manuals, 3035
Tutorials, 3036
arccos, function, 586
arcsin, function, 586
arctan, function, 586
arctan2, function, 586
arithmetic operations, 581, 768
  addition, 581
  division, 581
  exponentiation, 581
  multiplication, 581
  subtraction, 581
ask, tool, 2657
assigned, reference type, 750
assignment
  conditional, 662
  indexed, 578
  issues with controlling indices, 581
  over subsets, 580
  scalar, 578
  sparse assignments, 663
  statement, 577
  using labels explicitly, 580
asterisk
  in set definitions, 543
  marking errors, 756, 766
  use in comments, 538
attributes, set, 552
automated tuning
  GUROBI, 1747
BARON
  complementarity constraints, 1421
  global optimization, 1414
  HS, 1420
  irreducibly inconsistent system(IIS), 1420
  logfile, 1416
  model statuses, 1417
  solution pool, 1419
  solver statuses, 1417
  termination messages, 1417
  variable bounds, 1415
BARON, solver, 1414
barrier algorithm
  CPLEX, 1570
  GUROBI, 1743
barrier method
  CPLEX, 1570
BCH, examples
  Solver Usage, 1050
Benders’ Decomposition
  DECIS, 1669
best practice, approximating non-smooth functions
  DICOPT, 1698
best practice, avoiding function domain errors
  CONOPT, 1466
  DICOPT, 1699
best practice, avoiding function evaluation errors
  CONOPT, 1466
  DICOPT, 1699
best practice, formulating a good model
  CONOPT, 1468
  best practice, good starting point
    CONOPT, 1468
    DICOPT, 1696
  best practice, intermediate variables
    CONOPT, 1470
  best practice, introducing slack variables
    DICOPT, 1716
  best practice, model debugging
    DICOPT, 1717
  best practice, scaling
    DICOPT, 1696
  best practice, scaling
    CONOPT, 1471
  best practice, scaling intermediate variables
    CONOPT, 1472
  best practice, scaling with GAMS scale option
    CONOPT, 1473
  best practice, setting bounds
    DICOPT, 1696
  best practice, setting bounds
    CONOPT, 1469
  best practice, simple expressions
    CONOPT, 1470
  best practice, small big M
    DICOPT, 1698
  best practice, solving the relaxed model first
    DICOPT, 1696
  best practice, stopping criteria
    DICOPT, 1715
  best practice, using different MIP solvers
    DICOPT, 1716
  best practice, using different NLP solvers
    DICOPT, 1716
beta distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
beta, function, 586
betaReg, function, 586
bilevel programs
  EMP, 1207
  JAMS, 1893
binary, keyword, 532
binomial distribution
  GAMS Stochastic Library, 1137
  LINDO Sampling Library, 1141
binomial, function, 586
bm suffixes, put-file, 996
bool_and, function, 592
bool_eqv, function, 592
bool_imp, function, 592
bool_not(x), function, 592
bool_or, function, 592
bool_xor, function, 592
boolean operations, 666
branch and bound global optimization
  ANTIGONE, 1413
branch-and-bound algorithm
SBB, 2239
branch-and-cut algorithm
CPLEX, 1570
branch-and-cut optimizer
MOSEK, 2086
Branch-and-Cut-and-Heuristic facility (BCH)
Solver Usage, 1046
branching priority value, variable (.prior), 616
break, keyword, 532
capability problems, solver status, 761
card, function, 590
card, keyword, 532
card, operator on sets, 571
case
suffixes, put-file, 996
cauhcy distribution
GAMS Stochastic Library, 1136
LINDO Sampling Library, 1140
CBC, 1429
CBC, solver, 1429
cc
suffixes, put-file, 996
cdfBVN, function, 1146
cdfTVN, function, 1146
cdfUVN, function, 1146
ceil, function, 586
certainty, function, 586
CFG API, expert-level API, 3041
chance constraints, 1242
DE, 1656
CHENERY, example from GAMSlib, 621, 667
chiSquare distribution
GAMS Stochastic Library, 1136
LINDO Sampling Library, 1140
chk4upd, tool, 2668
cholesky, tool, 2669
CNS, model type, 637
collections of models
GUSS, 1804
column listing, GAMS output, 756
comma
in data lists, 538
in put statements, 991
command line parameters, list, 792
comment
using $eolcom, 539
using $inlinecom, 539
compilation
errors, 766
errors at ... time, 767
output, 747
compile-time constant, 789
compile-time variable, 786
complement, set operation, 562
complementarity constraints
BARON, 1421
componentEDate, predefined symbol, 599
ComponentLicenseOptions, predefined symbol, 599
componentMDate, predefined symbol, 599
compress, 982
compute server
GUROBI, 1747
concauity
ANTIGONE, 1412
concurrent optimizer
CPLEX, 1570
GUROBI, 1748
conditional compilation, 969
conditional expressions
numerical values, 661
operator precedence, 661
using set membership, 659
with logical operators, 658
with numerical relationship operators, 656
confidence interval
DECIS, 1668
Conflict refiner
CPLEX, 1570
CONOPT
best practice, good starting point, 1468
best practice, avoiding function domain errors, 1466
best practice, avoiding function evaluation errors, 1466
best practice, formulating a good model, 1468
best practice, intermediate variables, 1470
best practice, scaling, 1471
best practice, scaling intermediate variables, 1472
best practice, scaling with GAMS scale option, 1473
best practice, setting bounds, 1469
best practice, simple expressions, 1470
CONOPT algorithm, 1479
CONOPT algorithm, crash procedure, 1487
CONOPT algorithm, definition equations, 1488
CONOPT algorithm, initial feasible solution, 1490
CONOPT algorithm, linear mode, 1491
CONOPT algorithm, loss of feasibility, 1499
CONOPT algorithm, nonlinear mode, 1491
CONOPT algorithm, pre-processing, 1481
CONOPT algorithm, preprocessing and function evaluation errors, 1488
CONOPT algorithm, scaling, 1489
CONOPT algorithm, SLP procedure, 1492
CONOPT algorithm, SQP procedure, 1493
CONOPT algorithm, starting point, 1481
CONOPT algorithm, steepest edge procedure, 1493
constrained nonlinear system (CNS), 1498
diagnostics, 1463
DNLP models, problems, 1475
DNLP models, reformulations, 1475
DNLP models, smooth approximations, 1476
Cutting Plane
- AlphaECP, 1387, 1391
cvPower, function, 586
data
- entered as parameters, 603
- entered as tables, 606
- entry, 601
- manipulations with parameters, 577
- type, 529
Data Envelopment Analysis (DEA) with GUSS
- GUSS, 1814
data exchange
- DB2, 449
- MS Access, 451
- MySQL, 458
- Oracle, 462
- SQL Server, 465
- SyBase, 470
data exchange, ACCESS
- MDB2GMS, 2906
data exchange, EXCEL
- GDX2XLS, 2781
- XLS2GMS, 3016
- XLSDump, 3032
data exchange, GAMS
- GDX, 1162
data transformation
- CHOLESKY, 2669
- CSDP, 2670
- EIGENVALUE, 2686
- EIGENVECTOR, 2687
- INVERT, 2904
databases, data exchange, 449
DCT API, expert-level API, 3041
DE
- chance constraints, 1656
- deterministic equivalent, 1654
- expected value problem (EVP), 1656
- logfile, 1660
- probability distributions, continuous, 1655
- probability distributions, discrete, 1655
- probability distributions, joint, 1654
- random variables, 1654
- reformulation techniques, 1657
- reformulation, chance constraints, 1659
- reformulation, CVaR, 1660
- reformulation, VaR, 1659
- risk measures, 1656
- sampling, 1655
- scenario tree, 1654
- scenarios, 1654
- stages, 1654
- stochastic linear program, two-stage, 1657
- stochastic programming, 1653
- stochastic programming, EMP, 1653
- DE, solver, 1653
- DEA modeling with GUSS
- GUSS, 1814
declaration
- acronym, 611
- decimals, global option, 675
DECIS
- Benders’ Decomposition, 1669
- confidence interval, 1668
- core model, 1670
- CPLEX as optimizer, 1679
- decision stages, 1672
- deterministic equivalent, 1666
- error messages, 1690
- example models, 1685
- expected value problem, 1668
- MINOS as optimizer, 1678
- Monte-Carlo pre-sampling, 1669
- Monte-Carlo sampling, 1668
- output, 1680
- probability distributions, 1672
- random parameter, 1667
- regularization, 1669
- stochastic linear program, 1666
- stochastic linear programs, two-stage, 1666
- stochastic parameters, dependent, 1675
- stochastic parameters, independent, 1672
- Stochastic programming, 1666
- Stochastic Programming, EMP, 1666
- uncertainty, 1667
- universe problem, 1668
DECIS, solver, 1665
decision stages
- DECIS, 1672
declaration
- alias, 544
- equation, 623
- file, 988
- function, 1134
- model, 634
- parameter, 603
- scalar, 602
- set, 541
- statements, 527
- table, 605
- variable, 612
decompress, 982
defined, reference type, 750
definition
- of a model, 634
- of data, 529
- of scalars, 602
- of symbols, 531
- statements, 527
deterministic equivalent
- DE, 1654
- DECIS, 1666
diag, 660
diag, keyword, 532
diag, predefined symbol, 599
diagnostic tags
IPOPT, 1831
diagnostics
CONOPT, 1463
DICOPT
  best practice, approximating non-smooth functions, 1698
  best practice, avoiding function domain errors, 1699
  best practice, avoiding function evaluation errors, 1699
  best practice, good starting point, 1696
  best practice, introducing slack variables, 1716
  best practice, model debugging, 1717
  best practice, scaling, 1696
  best practice, setting bounds, 1696
  best practice, small big M, 1698
  best practice, solving the relaxed model first, 1696
  best practice, stopping criteria, 1715
  best practice, using different MIP solvers, 1716
  best practice, using different NLP solvers, 1716
DICOPT algorithm, 1694
DICOPT algorithm
DICOPT, 1694
DICOPT, solver, 1693
dictionary file
  CONVERT, 1555
difference, set operation, 562
direction of optimization, 649
discrete variables
details, 727
in models, 636
overview, 613
disjunctive programming
  EMP, 1212
  JAMS, 1904
display
d controls local, 676
d example, 672
generating data in list format, 678
global controls, 675
introduction, 671
label order, 673
syntax, 671
display control, 702
display, keyword, 533
distributed optimization
  GUROBI, 1748
distributions
  beta, 1136, 1140
  binomial, 1137, 1141
  cauchy, 1136, 1140
  chiSquare, 1136, 1140
  exponential, 1136, 1140
  f, 1136, 1140
gamma, 1136, 1140
gumbel, 1136, 1140
hyperGeo, 1137, 1141
invGaussian, 1136
laplace, 1136, 1140
logarithmic, 1137, 1141
logistic, 1136, 1140
logNormal, 1136, 1140
negBinomial, 1137, 1141
normal, 1136, 1140
pareto, 1136, 1140
poisson, 1137, 1141
rayleigh, 1136
studentT, 1136, 1140
triangular, 1136, 1140
uniform, 1136, 1140
uniformInt, 1137
weibull, 1136, 1140
div, function, 586
div0, function, 586
DNLP models, problems
  CONOPT, 1475
DNLP models, reformulations
  CONOPT, 1475
DNLP models, smooth approximations
  CONOPT, 1476
DNLP models, smooth examples
  CONOPT, 1477
DNLP, model type, 636
do, keyword, 533
dollar condition, 655
control over the domain of definition, 667
dollar operator, 655
e xample, 656
in indexed operations, 665
nested, 662
on the left, 663
on the right, 664
with dynamic sets, 563
within the algebra, 666
dollar control option
  Detailed Descriptions, 878
  Introduction, 871
  Lists, 872
  Syntax, 871
dollar control options, 871
  $abort, 878
  $abort.noError, 878
  $batInclude, 879
  $call, 880
  $call.Async, 881
  $call.AsyncNC, 881
  $call.checkErrorLevel, 882
  $clear, 882
  $clearError, 883
  $clearErrors, 883
  $comment, 883
  $compress, 884
  $decompress, 884
  $dollar, 885
$\text{double}, 885
$\text{drop}, 886
$\text{dropEnv}, 886
$\text{dropGlobal}, 886
$\text{dropLocal}, 887
$\text{echo}, 887
$\text{echoN}, 887
$\text{eject}, 888
$\text{else}, 888
$\text{elseif}, 889
$\text{elseifE}, 889
$\text{elseifI}, 889
$\text{encrypt}, 889
$\text{endif}, 890
$\text{eolCol}, 890
$\text{error}, 891
$\text{escape}, 891
$\text{eval}, 893
$\text{eval.Set}, 893
$\text{evalGlobal}, 895
$\text{evalGlobal.Set}, 895
$\text{evalLocal}, 895
$\text{evalLocal.Set}, 895
$\text{exit}, 896
$\text{expose}, 896
$\text{funcLibIn}, 896
$\text{gdxIn}, 896
$\text{gdxLoad}, 897
$\text{gdxOut}, 897
$\text{gdxUnload}, 897
$\text{goto}, 898
$\text{hidden}, 898
$\text{hiddenCall}, 899
$\text{hiddenCall.Async}, 899
$\text{hiddenCall.AsyncNC}, 899
$\text{hiddenCall.checkErrorLevel}, 899
$\text{hide}, 899
$\text{if}, 900
$\text{ifE}, 900
$\text{ifThen}, 900
$\text{ifThenE}, 901
$\text{ifThenI}, 902
$\text{if}, 903
$\text{include}, 904
$\text{inlineCom}, 904
$\text{kill}, 904
$\text{label}, 904
$\text{libinclude}, 904
$\text{lines}, 905
$\text{load}, 905
$\text{loadDC}, 907
$\text{loadDCM}, 908
$\text{loadDCR}, 908
$\text{loadFiltered}, 909
$\text{loadFilteredM}, 909
$\text{loadFilteredR}, 909
$\text{loadM}, 910
$\text{loadR}, 911
$\text{log}, 912
$\text{macro}, 912
$\text{maxCol}, 913
$\text{maxGoto}, 913
$\text{minCol}, 913
$\text{offCheckErrorLevel}, 914
$\text{offDelim}, 914
$\text{offDigit}, 915
$\text{offDollar}, 916
$\text{offDotL}, 916
$\text{offDotScale}, 917
$\text{offECImplicitLoad}, 919
$\text{offEcho}, 917
$\text{offEchoS}, 917
$\text{offEchoV}, 917
$\text{offEmbedded}, 919
$\text{offEmbeddedCode}, 920
$\text{offEmpty}, 922
$\text{offEnd}, 923
$\text{offEolCom}, 924
$\text{offEps}, 924
$\text{offEpsToZero}, 926
$\text{offExpand}, 927
$\text{offExternalInput}, 927
$\text{offExternalOutput}, 929
$\text{offFiltered}, 930
$\text{offGlobal}, 930
$\text{offIDCProtect}, 931
$\text{offInclude}, 931
$\text{offInline}, 932
$\text{offListing}, 932
$\text{offLocal}, 933
$\text{offLog}, 934
$\text{offMacro}, 935
$\text{offMargin}, 935
$\text{offMulti}, 936
$\text{offNestCom}, 939
$\text{offOEIXRef}, 946
$\text{offOrder}, 939
$\text{offPut}, 940
$\text{offPutS}, 940
$\text{offPutV}, 940
$\text{offRecurse}, 942
$\text{offStrictSingleton}, 943
$\text{offSuffixDLVars}, 943
$\text{offSymList}, 944
$\text{offSymXRef}, 945
$\text{offText}, 945
$\text{offUElList}, 946
$\text{offUNDF}, 947
$\text{offUni}, 947
$\text{offUpper}, 941
$\text{offVerbatim}, 948
$\text{offWarning}, 949
$\text{offCheckErrorLevel}, 914
$\text{offDelim}, 914
$\text{offDigit}, 915
$\text{offDollar}, 916
INDEX

SonDotL, 916
SonDotScale, 917
SonECImplicitLoad, 919
SonEcho, 917
SonEchoS, 917
SonEchoV, 917
SonEmbedded, 919
SonEmbeddedCode, 920
SonEmpty, 922
SonEnd, 923
SonEolCom, 924
SonEps, 924
SonEpsToZero, 926
SonExpand, 927
SonExternalInput, 927
SonExternalOutput, 929
SonFiltered, 930
SonGlobal, 930
SonIDCProtect, 931
SonInclude, 931
SonInline, 932
SonListing, 932
SonLocal, 933
SonLog, 934
SonMacro, 935
SonMargin, 935
SonMulti, 936
SonMultiR, 937
SonNestCom, 939
SonOEIXRef, 946
SonOrder, 939
SonPut, 940
SonPutS, 940
SonPutV, 940
SonRecurse, 942
SonStrictSingleton, 943
SonSuffixDLVars, 943
SonSymList, 944
SonSymXRef, 945
SonText, 945
SonUEIList, 946
SonUNDF, 947
SonUni, 947
SonUpper, 941
SonVerbatim, 948
SonWarning, 949
$prefixPath, 951
$protect, 952
$purge, 952
$remark, 952
$title, 963
$scratchFileName, 953
#set, 953
#setArgs, 954
#setComps, 955
#setDDLList, 955
#setEnv, 956
#setGlobal, 957
#setLocal, 957
#setNames, 957
#shift, 959
#show, 959
#showFiles, 961
#showMacros, 961
#showVariables, 961
#single, 961
#splitOption, 962
#stars, 963
#stop, 963
#sysInclude, 964
#terminate, 964
#title, 964
#unload, 965
#use205, 967
#use225, 968
#use999, 968
#version, 968
#warning, 969
$phantom, 950
dollar operator, 655
domain checking, 554
domain restriction condition, 667
dot
  in level and marginal listings, 763
  in many to many mappings, 547
  in parameter definition, 605
  in set definition, 546
  in sets, 546
  in tables, 608
double dash parameters, 784
dual simplex algorithm
  CPLEX, 1570
  GUROBI, 1743
dual value, variable (.m), 616
dynamic set
  assigning membership to, 558
  assigning membership to singleton sets, 561
  dollar assignments, 563
  equations defined over the domain of, 560
  example, 559
  in equations, 566
  indexed operations, 564
  introduction, 558
  syntax, 559
  using dollar controls with, 563
  with multiple indices, 560
e-format, 675
echo print, GAMS output, 747
ECP
  AlphaECP, 1387
edge-concavity
  ANTIGONE, 1412
edge-convexity
  ANTIGONE, 1412
eDist, function, 587
eigenvalue, tool, 2686
eigenvector, tool, 2687
else, keyword, 533
elseif, keyword, 533
Embedded Code
Building your own Embedded Python Code
Library, 1090
Concept, 1069
Connect, 1091
Encodings, 1084
Exchange via Files and Environment Variables, 1082
Extending GMSPython, 1089
Motivation, 1069
Multiple Independent Python Sessions, 1084
Performance Considerations of Embedded Python Code, 1088
Python, 1076
Simple Example, 1070
Simple Example Connect, 1072
Simple Example Python, 1070
Syntax, 1074
Troubleshooting Embedded Python Code, 1085
Using an External Python 3 Installation, 1090
Using the Object-oriented API, 1081
embedded complementarity systems
EMP, 1199
JAMS, 1897
embeddedHandle, function, 594
EMP, 1184
bilevel programs, 1207
DE, 1653
disjunctive programming, 1212
embedded complementarity systems, 1199
EMP annotations, 1185
EMP keywords, list of, 1251
equilibrium problems, 1193
equilibrium problems with shared constraints, 1201
equilibrium problems with shared variables, 1203
JAMS, 1888
quasi-variational inequalities (QVI), 1191
soft constraints, 1186
stochastic programming, 1217
variational inequalities (VI), 1188
EMP Stochastic Programming, 1217
DE, 1653
DECIS, 1666
EMP, model type, 637
emphasis settings
SCIP, 2250
empinfo file
EMP, 1185
JAMS, 1905
encrypt, 983
end of line, 537, 606
endecrypt, tool, 2689
endfor, keyword, 533
endif, keyword, 533
endloop, keyword, 533
endogenous, 629
endwhile, keyword, 533
ENLP
JAMS, 1899
ENLP, example
JAMS, 1902
entropy, function, 587
environment variables in GAMS, 788
eps
definition, 598
marginal value, 763
used with variables, 763
eps, keyword, 533
eq, keyword, 535
equation, 623
attributes, 631
declaration, 623
definition, 624
functions in equation definitions, 629
indexed equations, 626
listing, 754
logic equations, 628
scalar equations, 626
suffixes, 631
types, 625
equation, keyword, 533
equations, keyword, 533
equilibrium problems with shared constraints, EMP, 1201
equilibrium problems with shared variables, EMP, 1203
equilibrium problems, EMP, 1193
eqv, keyword, 535
error
arithmetic errors, 394
debugging, 402
exceeding GAMS limits, 395
finding compilation errors, 377
finding execution errors, 393
fixing compilation errors, 379
fixing execution errors, 393
handling, 598
list of common compilation errors, 379
messages, 334
put errors, 1030
reporting, 765
reporting compilation errors, 766
reporting compilation time errors, 767
reporting execution errors, 768
reporting solve errors, 768
resolving model generation errors, 396
resolving solve errors, 398
using execError, 401
error internal solver failure, solver status, 761
error no solution, model status, 760
error setup failure, solver status, 761
INDEX

error solver failure, solver status, 761
error system failure, solver status, 761
error unknown, model status, 760
errorf, function, 587
errorLevel, function, 594
ers
suffixes, put-file, 996
evaluation interrupt, solver status, 761
EXAMINER, solver, 1718
example models
DECS, 1685
EXCEL, data exchange
GDX2XLS, 2781
XLS2GMS, 3016
XLSDump, 3032
exception handling in equations, 666
exception, see dollar condition, 655
execError, function, 594
eexecSeed, function, 587
execute, keyword, 533
execute_load, keyword, 533
execute_loaddc, keyword, 533
execute_loadhandle, keyword, 533
execute_loadpoint, keyword, 533
execute_unload, example, 442
execute_unload, keyword, 533
execute_unloaddi, keyword, 533
execute_unloadidx, keyword, 533
exogenous, 629
exp, function, 587
expected value problem (EVP)
  DE, 1656
  DECS, 1668
expert-level APIs, 3041
  CFG API, 3041
  DCT API, 3041
  GDX API, 3041
  GEV API, 3041
  GMO API, 3041
  IDX API, 3042
  OPT API, 3042
  PAL API, 3042
explanatory text, 613
exponent, 582
exponential distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
expression tree reformulation
  ANTIGONE, 1412
extended arithmetic, 598
Extended Cutting Plane
  AlphaECP, 1387, 1390, 1391
Extended Mathematical Programming, see EMP, 1184
extended range values, 598
eps, 598
inf, 598
na, 598
undf, 598
external equations, 1148
CONOPT, 1502
extrinsic functions
  build your own library, 1145
  CONOPT, 1502
  CPP Library, 1146
  Fitpack Library, 1134
  introduction, 1133
  LINDO Sampling Library, 1139
  Mutex Library, 1142
  Piecewise Polynomial Library, 1135
  Stochastic Library, 1136
  using function libraries, 1133
  vs. external equations, 1147
f distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
fact, function, 587
Farkas certificate
  MOSEK, 2088
feasible relaxation
  CPLEX, 1571
  GUROBI, 1746
feasible solution, model status, 760
FERTD, example from GAMSlib, 665
file
  defining, 988
  statement, 988
file summary, GAMS output, 765
file, keyword, 533
files, keyword, 533
first order optimality conditions
  JAMS, 1891
fitFunc, function, 1135
fitParam, function, 1135
floor, function, 587
for
  example, 688
  statement, 687
  syntax, 688
for, keyword, 533
frac, function, 587
free, keyword, 533
function, 584
  extrinsic, 1133
  intrinsic, 584
function suffixes, see suffixes, function, 596
function, keyword, 533
functions, 584
  abs, 586
  arccos, 586
  arcsin, 586
  arctan, 586
  arctan2, 586
  beta, 586
  betaReg, 586
  binomial, 586
bool_and, 592
bool_eqv, 592
bool_imp, 592
bool_not(x), 592
bool_or, 592
bool_xor, 592
card, 590
cdfBVN, 1146
cdfTVN, 1146
cdfUVN, 1146
ceil, 586
centropy, 586
cos, 586
cosh, 586
cosine, 1146
cvPower, 586
div, 586
div0, 586
eDist, 587
embeddedHandle, 594
extropy, 587
errorf, 587
errorLevel, 594
execError, 594
execSeed, 587
exp, 587
fact, 587
fitFunc, 1135
fitParam, 1135
floor, 587
frac, 587
gamma, 587
gammaReg, 587
gamsRelease, 594
gamsVersion, 594
gday, 593
gdow, 593
ghour, 593
gLeap, 593
gmillisec, 593
gminute, 593
gmonth, 593
gsecond, 593
gyear, 593
handleCollect, 594
handleDelete, 594
handleStatus, 594
handleSubmit, 594
heapFree, 594
heapLimit, 594
heapSize, 594
iFThen, 592
jdate, 593
jnow, 593
jobHandle, 594
jobKill, 595
jobStatus, 595
jobTerminate, 595
jstart, 593
jtime, 593
licenseLevel, 595
licenseStatus, 595
log, 587
log10, 587
log2, 587
logBeta, 587
logGamma, 587
logit, 587
mapVal, 595
max, 587
maxExecError, 595
min, 587
mod, 588
ncpCM, 588
ncpF, 588
ncpVUpow, 588
ncpVUsin, 588
normal, 588
numCores, 595
ord, 590
pdfBVN, 1146
pdfTVN, 1146
pdfUVN, 1146
pi, 588, 1146
platformCode, 596
poly, 588
power, 588
pwpFunc, 1136
randBinomial, 588
randLinear, 588
randTriangle, 588
readyCollect, 596
rel_eq, 592
rel_ge, 592
rel_gt, 592
rel_le, 592
rel_lt, 592
rel_ne, 593
round, 588
rPower, 589
setMode, 1146
sigmoid, 589
sign, 589
signPower, 589
sin, 589
sine, 1146
sinh, 589
sleep, 596
slexp, 589
slog10, 589
srec, 589
sqexp, 589
sqlog10, 589
sqrt, 589
sqrec, 590
sqr, 590
tan, 590

tanh, 590

timeClose, 596

timeComp, 596

timeElapsed, 596

timeExec, 596

timeStart, 596

trunc, 590

uniform, 590

uniformInt, 590

vcPower, 590

functions, keyword, 533

fx
suffixes, variable, 616

gamma distribution

GAMS Stochastic Library, 1136

LINDO Sampling Library, 1140

gamma, function, 587

gammaReg, function, 587

GAMS call

Introduction, 775

specifying options, 777

specifying options with spaces in value, 777

GAMS command line parameters, 792

GAMS dot options for solvers

Solver Usage, 1036

GAMS language items, 530

characters, 531

comments, 538

delimiters, 538

identifiers, 535

keywords, 532

labels, 536

Numbers, 537

reserved words, 532

text, 537

GAMS log, 772

GAMS options for solvers

Solver Usage, 1034

GAMS output, 746

column listing, 756

compilation output, 747

customizing, 769

echo print, 747

equation listing, 754

error reporting, 765

example, 746

execution output, 753

file summary, 765

final execution summary, 764

include file summary, 752

listing file, 746

model generation output, 754

model statistics, 757

model status, 759

output file, 746

range statistics, 757

report summary, 764

solution listing, 762

solution report, 758

solve summary, 758

solver report, 761

solver status, 761

symbol listing map, 751

symbol reference map, 749

unique element listing map, 751

GAMS Studio, Tool, 2691

GAMSCHK

*.GCK file, 1723

*.GCK file, comments, 1726

*.GCK file, page width, 1726

cheatsheet, 1741

known bugs, workaround, 1734

nonlinear terms, 1726

options file, 1732

procedures, model schematic for components (PICTURE), 1730

procedures, diagnosing infeasible models (NONOPT), 1731

procedures, diagnosing unbounded models (NONOPT), 1731

procedures, displaying coefficients (DISPLAYCR), 1727

procedures, listing characteristics of items per block (BLOCKLIST), 1729

procedures, model schematic per block (BLOCKPIC), 1729

procedures, post optimality computations (POSTOPT), 1730

procedures, retrieving characteristics of items (MATCHIT), 1728

procedures, summary information on scaling per block (BLOCKPIC), 1729

procedures, testing bounds (ADVISORY), 1731

procedures, testing for specification errors (ANALYSIS), 1729

reserved names, 1740

GAMSCHK, solver, 1722

GAMSlib, Model Libraries, 1255

gamsRelease, function, 594

gamsVersion, function, 594

gday, function, 593

gdow, function, 593

GDX

read gdx file, 1162

viewer, 2697

write gdx file, 1175

GDX API, expert-level API, 3041

GDX Viewer, GAMS Studio, 2697

gdx2access, tool, 2755

gdx2har, tool, 2764

gdx2sqlite, tool, 2773

gdx2veda, tool, 2778

GDX2XLS

colors, 2784

custom formats, 2784
example, detailed, 2781
examples, 2785
filter, 2782
ini file, 2783
options, 2783
settings, 2783
GDX2XLS, tool, 2781
gdxcopy, tool, 2788
gdxdiff, tool, 2790
gdxmerge, tool, 2793
gdxrmerge, tool, 2803
gdxrank, tool, 2807
gdxrrow, tool, 2843
gdxrename, tool, 2841
gdxrrew, tool, 2866
gf, keyword, 535
gE, keyword, 535
gE, distribution, 1137
GEV API, expert-level API, 3041
ghour, function, 593
gleap, function, 593
global optimization
BARON, 1414
LINDO, 1962
global solver
LINDO, 1962
gmillisec, function, 593
gminute, function, 593
GMO API, expert-level API, 3042
gmonth, function, 593
grad
sufixes, function, 597
gradH
sufixes, function, 597
gradL
sufixes, function, 597
gradn
sufixes, function, 597
GRG algorithm
CONOPT, 1479
grid computing, 715
gsecond, function, 593
gt, keyword, 535
GTM, example from GAMSlib, 665, 667
gumbel distribution
GAMS Stochastic Library, 1136
LINDO Sampling Library, 1140
GUROBI
automated tuning, 1747
barrier algorithm, 1743
compute server, 1747
concurrent optimizer, 1748
distributed optimization, 1748
dual simplex algorithm, 1743
feasible relaxation, 1746
GUROBI-Link, 1801
IIS, 1743
Irreducibly inconsistent set of constraints (IIS), 1743
parameter tuning, 1747
sensitivity analysis, 1743
starting solution for MIPs, 1744
GUROBI, solver, 1743
GUROBI-Link
GUROBI, 1801
GUSS
Data Envelopment Analysis (DEA) with GUSS, 1814
collections of models, 1804
cross validation, 1808
cross validation, ten-fold, 1808
cross validation, ten-fold, example, 1808
DEA modeling with GUSS, 1814
model instance, 1804
options in dictionary set, 1806
quadratic programming, example, 1812
set dictionary, 1804
subsolvers not compatible with GUSS, 1804
symbol mapping information, 1804
updating parts of a model iteratively, 1804
GUSS, solver, 1804
gyear, function, 593
handleCollect, function, 594
handleDelete, function, 594
handleStatus, function, 594
handleSubmit, function, 594
har2gdx, tool, 2764
hdcc
sufixes, put-file, 996
hdcr
sufixes, put-file, 997
hdll
sufixes, put-file, 997
heapFree, function, 594
heapLimit, function, 594
heapSize, function, 594
hess
sufixes, function, 597
hessn
sufixes, function, 597
high
sufixes, function, 597
HIGHS, solver, 1820
hyperGeo distribution
GAMS Stochastic Library, 1137
LINDO Sampling Library, 1141
idecmds, tool, 2654
IDX API, expert-level API, 3042
if, keyword, 533
if-elseif-else
a statement, 680
example, 681
syntax, 681
INDEX 3287

ifThen, function, 592
IIS
  BARON, 1420
  CPLEX, 1570
  GUROBI, 1743
  LINDO, 1963
impl-asn, reference type, 750
index matching, 702
indexed operations, 582
  prod, 582
  sand, 582
  smax, 582
  smin, 582
  sor, 582
  sum, 582
imp, keyword, 535
impl-asn, reference type, 750
index matching, 702
indexed operations, 582
  prod, 582
  sand, 582
  smax, 582
  smin, 582
  sor, 582
  sum, 582
indices, controlling, 583
INDUS, example from GAMSlib, 609
inf
  extended range value, 598
  variable bound, 613
inf, keyword, 533
infeas
  suffixes, equation, 632
  suffixes, variable, 618
Infeasibility Finder
  CPLEX, 1570
infeasibility report
  MOSEK, 2088
infeasible, 756, 763
infeasible - no solution, model status, 760
infeasible, model status, 760
infeas, solution marker, 763, 764
initial values, 527, 619, 621
initialization
  of data, 601
  of parameters, 603
integer
  variable, 618
  variables, 636
integer infeasible, model status, 760
integer solution, model status, 760
integer, keyword, 533
inner-point optimizer
  MOSEK, 2086
intermediate infeasible, model status, 760
intermediate non-integer, model status, 760
intersection, set operation, 562
intrinsic functions, 584
  GAMS utility and performance, 594
  logical, 591
  mathematical, 586
  string manipulation, 590
  time and calendar, 593
invert, tool, 2904
invGaussian distribution
GAMS Stochastic Library, 1136
IPOPT, solver, 1826
IPOPTO
  diagnostic tags, 1831
  logfile, 1828
Irreducibly inconsistent set of constraints (IIS)
  CPLEX, 1570
  GUROBI, 1743
irreducibly inconsistent system (IIS)
  BARON, 1420
iteration
  default limit, 759
iteration interrupt, solver status, 761
IUS
  LINDO, 1963
JAMS
  bilevel programs, 1893
  disjunctive programming, 1904
  embedded complementarity systems, 1897
  EMP, 1888
  empinfo file, 1905
  ENLP, 1899
  ENLP, example, 1902
Extended Mathematical Programming, 1888
  first order optimality conditions, 1891
  KKT conditions, 1891
  Lagrangian, 1891
  Lagrangian, extended form, 1901
  LogMIP, 1904
  modeling discrete choices, 1904
  MOPEC, 1899
Multiple Optimization Problems with Equilibrium Constraints (MOPECs), 1899
  multiple optimizing agents, 1899
  Nash Games, 1899
  penalization, 1899
  penalty, quadratic, 1900
  reformulation, 1888
  reformulation, bilevel as MPCC, 1893
  reformulation, ENLP as MCP, 1891
  reformulation, ENLP as NLP, 1891
  reformulation, ENLP as NLP, theory, 1903
  reformulation, ENLP as VI, 1901
  reformulation, NLP as MCP, 1891
  reformulation, VI as complementarity problem, 1895
  scalar model, 1888
  subsolver, 1888
  variational inequalities (VIs), 1895
JAMS, solver, 1888
  jdate, function, 593
  jnow, function, 593
  jobHandle, function, 594
  jobKill, function, 595
  jobStatus, function, 595
  jobTerminate, function, 595
  jstart, function, 593
  jtime, function, 593
KESTREL, solver, 1906
keywords, 532
  abort, 532
  acronym, 532
  acronyms, 532
  alias, 532
  all, 532
  and, 532
  binary, 532
  break, 532
  card, 532
  continue, 532
  diag, 532
  display, 533
  do, 533
  else, 533
  elseif, 533
  endfor, 533
  endif, 533
  endloop, 533
  endwhile, 533
  eps, 533
  eq, 535
  equation, 533
  equations, 533
  eqv, 535
  execute, 533
  execute_load, 533
  execute_loaddc, 533
  execute_loadidx, 533
  execute_loadpoint, 533
  execute_unload, 533
  execute_unloadddi, 533
  file, 533
  files, 533
  for, 533
  free, 533
  function, 533
  functions, 533
  ge, 535
  gt, 535
  if, 533
  imp, 535
  inf, 533
  integer, 533
  le, 535
  logic, 533
  loop, 533
  lt, 535
  maximizing, 535
  minimizing, 535
  model, 533
  models, 533
  na, 533
  ne, 535
  negative, 533
  no, 533
  nonnegative, 533
  not, 533
  option, 533
  options, 533
  or, 533
  ord, 533
  parameter, 534
  parameters, 534
  positive, 534
  prod, 534
  put, 534
  put_utilities, 534
  putclear, 534
  putclose, 534
  putfncl, 534
  puthd, 534
  putheader, 534
  putpage, 534
  puttitle, 534
  puttl, 534
  repeat, 534
  sameas, 534
  sand, 534
  scalar, 534
  scalars, 534
  scenario, 535
  semicont, 534
  semiint, 534
  set, 534
  sets, 534
  singleton, 534
  smax, 534
  smin, 534
  solve, 534
  sor, 534
  sos1, 534
  sos2, 534
  sum, 534
  system, 534
  table, 534
  then, 534
  undf, 534
  until, 534
  using, 535
  variable, 534
  variables, 534
  while, 534
  xor, 534
  yes, 534
KKT conditions
  JAMS, 1891
KNITRO, solver, 1909
KORPET, example from GAMSlib, 607
l
  suffixes, equation, 631
  suffixes, variable, 617
label, 536
  in set definitions, 542
INDEX

in table definitions, 607
order data displays in list format, 679
order in multi-dimensional identifier displays, 673
quoted, 536
unquoted, 536
Lagrangian
  JAMS, 1891
Lagrangian, extended form
  JAMS, 1901
laplace distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
LaTeX
  MODEL2TEX, 2941
le
  suffixes, put-file, 997
le, keyword, 535
legal characters, 531
lic problem - no solution, model status, 760
licenseCodes, predefined symbol, 599
licenseLevel, function, 595
licenseStatus, function, 595
licensing problems, solver status, 761
limited memory
  CPLEX, 1588
LINDO
  global optimization procedure, 1962
  global solver, 1962
  IIS, 1963
  IUS, 1963
  linearization, 1962
  Nonlinear functions, supported, 1963
  Stochastic Programming, 2032
LINDO, solver, 1962
LINDOGLOBAL, solver, 1962
linearization
  AlphaECP, 1390, 1391
  LINDO, 1962
list
  of labels using Asterisks, 543
listing viewer, GAMS Studio, 2696
lj
  suffixes, put-file, 997
ll
  suffixes, put-file, 997
lm
  suffixes, put-file, 997
lo
  suffixes, equation, 631
  suffixes, variable, 616
locally infeasible, model status, 760
locally optimal, model status, 759
log, function, 587
log10, function, 587
log2, function, 587
logarithmic distribution
  GAMS Stochastic Library, 1137
LINDO Sampling Library, 1141
logBeta, function, 587
logfile
  ANTIGONE, 1403
  BARON, 1416
  DE, 1660
  IPOPT, 1828
  MOSEK, 2100
logGamma, function, 587
logic, keyword, 533
logical condition, 656
dollar operator, 655
logical operators, 658
mixed, 661
nested, 662
numerical expressions, 656
numerical relational operators, 657
numerical values, 661
set functions, 660
set membership, 659
logical operators, 658
logistic distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
logit, function, 587
LogMIP
  JAMS, 1904
logNormal distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
loop
  example, 683
  statement, 682
  syntax, 682
loop, keyword, 533
low
  suffixes, function, 597
lower bound, variable (.lo), 616
lp
  suffixes, put-file, 997
LP, model type, 636
lt, keyword, 535
lw
  suffixes, put-file, 997
m
  suffixes, equation, 631
  suffixes, variable, 617
Mac OS X Installation Notes, 299
macro, 977
mapping sets, 546
maps
  symbol listing, 751
  symbol reference, 749
  unique element listing (UEL), 751
mapVal, function, 595
MARCO, example from GAMSlib, 653
marginal, 763
marginal value, variable (.m), 616
max, function, 587
maxExecError, function, 595
maximizing, 649
maximizing, keyword, 535
mcfilter, tool, 2905
MCP, model type, 637
MDB2GMS
  $CALL command, 2937
  ACCESS, data exchange, 2906
  best practice, 2931
  command files, 2935
  command-line arguments, 2907
  example, multi-query batch, 2923
  example, multi-valued table, 2913
  example, single-valued table, 2912
  index mapping, 2920
  options, 2928
  requirements, 2906
  use, batch, 2906
  use, batch, multi-query, 2922
  use, interactive, 2925
mdb2gms, tool, 2906
MEXSS, example from GAMSlib, 1030
MILES, solver, 2033
min, function, 587
minimizing, 649
minimizing, keyword, 535
MINLP, model type, 637
MINOS as optimizer
  DECIS, 1678
MINOS, solver, 2053
MIP, model type, 637
MIQCP, model type, 637
mod, function, 588
model, 633
  attributes, 646
  library, 1255
  Limited domain for variables, 635
  syntax of statement, 633
  types, 636
model algebra
  CONVERT, 1555
model classification
  CNS, 637
  DNLP, 636
  EMP, 637
  LP, 636
  MCP, 637
  MINLP, 637
  MIP, 637
  MIQCP, 637
  MPEC, 637
  NLP, 636
  QCP, 636
  RMINLP, 637
  RMIP, 637
  RMIQCP, 637
  RMPEC, 637
model documentation
  MODEL2TEX, 2941
model instance
  GUSS, 1804
model library
  access, 1255
  explorer, 2742
model library explorer, GAMS Studio, 2742
model statistics, GAMS output, 757
model status, 759
  error no solution, 760
  error unknown, 760
  feasible solution, 760
  infeasible, 760
  infeasible - no solution, 760
  integer infeasible, 760
  integer solution, 760
  intermediate infeasible, 760
  intermediate non-integer, 760
  lic problem - no solution, 760
  locally infeasible, 760
  locally optimal, 759
  no solution returned, 760
  optimal, 759
  solved, 760
  solved singular, 760
  solved unique, 760
  unbounded, 759
  unbounded - no solution, 760
model statuses
  BARON, 1417
model, keyword, 533
MODEL2TEX
  example, 2943
  JSON file, 2943
  options, 2942
  usage, 2942
model2tex, tool, 2941
modeling discrete choices
  JAMS, 1904
models for which CONOPT is a good solver
  CONOPT, 1460
models, keyword, 533
Monte-Carlo pre-sampling
  DECIS, 1669
Monte-Carlo sampling
  DECIS, 1668
MOPEC
  JAMS, 1899
MOSEK
  branch-and-cut optimizer, 2086
  Farkas certificate, 2088
  infeasibility report, 2088
  interior-point optimizer, 2086
  logfile, 2100
  MOSEK-Link, 2138
  parallel optimization, 2087
  simplex optimizer, 2086
MOSEK, solver, 2086
MOSEK-Link
MOSEK, 2138
MPEC, model type, 637
mps2gms, tool, 2945
msappavail, tool, 2947
Multiple Optimization Problems with Equilibrium Constraints (MOPECs)
JAMS, 1899
multiple optimizing agents
EMP, 1193
JAMS, 1899
multiple solves, 653

na, extended range value, 598
na, keyword, 533
Nash Games
JAMS, 1899
ncpCM, function, 588
ncpF, function, 588
ncpVUpow, function, 588
ncpVUsin, function, 588
nd
suffices, put-file, 998
ne, keyword, 535
negative, keyword, 533
negBinomial distribution
GAMS Stochastic Library, 1137
LINDO Sampling Library, 1141
network optimizer
CPLEX, 1570
nj
suffices, put-file, 998
NLP models, non-smooth examples
CONOPT, 1478
NLP, model type, 636
NLPEC, solver, 2139
no solution returned, model status, 760
no, keyword, 533
non-default options, selecting
CONOPT, 1495
nonlinear
equations, 755
programming, 636
Nonlinear functions, supported
LINDO, 1963
nonnegative, keyword, 533
nopt, solution marker, 763
normal completion, solver status, 761
normal distribution
GAMS Stochastic Library, 1136
LINDO Sampling Library, 1140
normal, function, 588
not, keyword, 533
nr
suffices, put-file, 998
number of rows and columns in display, 675
numCores, function, 595
nw
suffices, put-file, 998
objective value, 759
objective variable, 649
OCTERACT, solver, 2151
ODHCPLEX, solver, 2163
OPT API, expert-level API, 3042
optimal, model status, 759
optimality gap, 1380
option
aggregation, 704
display control, 702
index matching, 702
introduction, 695
permutation, 706
projection, 704
syntax, 695
option, keyword, 533
options in dictionary set
GUSS, 1806
options, global
Solver Usage, 1034
options, keyword, 533
options, model-specific
Solver Usage, 1034
options, solver
Solver Usage, 1034
or, keyword, 533
ORANI, example from GAMSlib, 544
ord, function, 590
ord, keyword, 533
ordered set
card operator, 571
circular lag and lead operator, 574
introduction, 568
lags and leads in assignments, 572
lags and leads in equations, 575
linear lag and lead operator, 573
ord operator, 570
OsiCplex, solver, 2199
OsiGurobi, solver, 2199
OsiMosek, solver, 2199
OsiXpress, solver, 2199
output
DECIS, 1680
PAL API, expert-level API, 3042
parallel optimization
MOSEK, 2087
parameter, 601
custom evaluation, 609
dby assignment, 610
examples, 603
higher dimensions, 604
statement, 602
syntax, 603
parameter tuning
G U R O B I ,  1 7 4 7
parameter, formats, 6 0 1
parameter, keyword, 5 3 4
parameters, keyword, 5 3 4
parameters, see parameter, 6 0 1
pareto distribution
  G A M S Stochastic Library, 1 1 3 6
  L I N D O Sampling Library, 1 1 4 0
PATH, solver, 2 2 0 5
PATHNLP, solver, 2 2 0 2
PE
  suffixes, put-file, 9 9 9
PD F
  M O D E L 2 T E X , 2 9 4 1
  pdfBV N , function, 1 1 4 6
  pdfTV N , function, 1 1 4 6
  pdfUV N , function, 1 1 4 6
pdir
  suffixes, put-file, 9 9 8
penalization
  J A M S , 1 8 9 9
penalty, quadratic
  J A M S , 1 9 0 0
permutation, 7 0 6
pi, function, 5 8 8 , 1 1 4 6
platformCode, function, 5 9 6
poisson distribution
  G A M S Stochastic Library, 1 1 3 7
  L I N D O Sampling Library, 1 1 4 1
poly, function, 5 8 8
positive, keyword, 5 3 4
posix, tool, 2 9 4 8
post-optimality analysis
  C P L E X , 1 5 9 1
power, function, 5 8 8
precision, fixed, 7 6 3
predefined symbols
  componentEDate, 5 9 9
  ComponentLicenseOptions, 5 9 9
  componentMDate, 5 9 9
diag, 5 9 9
licenseCodes, 5 9 9
sameAs, 5 9 9
solverCapabilities, 5 9 9
solvers, 5 9 9
sortedUels, 5 9 9
prior
  suffixes, variable, 6 1 7
priorities for branching
    example, 7 3 2
    introduction, 7 3 2
probability distributions
  DECIS, 1 6 7 2
probability distributions, continuous
  DE, 1 6 5 5
probability distributions, discrete
  DE, 1 6 5 5
probability distributions, joint
  DE, 1 6 5 4
problem types, 6 3 6
prod, keyword, 5 3 4
prod, operator, 5 8 2
product operator
  in set definition, 5 4 7
  in sets, 5 4 7
projection, 7 0 4
PROLOG, example from G A M S lib, 6 3 4 , 6 5 2
ps
  suffixes, put-file, 1 0 0 0
pseudo costs
  S B B , 2 2 3 9
pseudo-convex
  AlphaECP, 1 3 8 7 , 1 3 9 1 , 1 4 0 1
put, 4 4 0
  appending to a file, 9 9 4
  closing a file, 9 9 3
  command line parameters, 1 0 1 6
cursor control, 1 0 0 5
defining files, 9 8 8
effects, 1 0 3 0
deexmples, 9 9 0 , 1 0 1 9
exception handling, 1 0 2 9
global item formatting, 1 0 1 7
local item formatting, 1 0 2 1
numeric items, 1 0 1 2
output items, 1 0 0 8
page format, 1 0 0 4
page sections, 1 0 0 2
paging control, 1 0 0 4
positioning the cursor on a page, 1 0 0 5
set value items, 1 0 1 4
syntax, 9 8 8
system suffixes, 1 0 1 5
text items, 1 0 0 8
put current cursor control
  .cc, 1 0 0 6
  .cr, 1 0 0 7
  .hdcc, 1 0 0 6
  .hdcr, 1 0 0 7
  .tlcc, 1 0 0 6
  .tlcr, 1 0 0 7
put, keyword, 5 3 4
put_Utilities, keyword, 5 3 4
putclear, keyword, 5 3 4
putclose, keyword, 5 3 4
putfnc1, keyword, 5 3 4
putlhd, keyword, 5 3 4
putheader, keyword, 5 3 4
putpage, keyword, 5 3 4
puttitle, keyword, 5 3 4
puttl, keyword, 5 3 4
pw
  suffixes, put-file, 1 0 0 0
pwpFunc, function, 1 1 3 6
QCP, model type, 6 3 6
quadratic programming, example
INDEX 3293

GUSS, 1812
quasi-variational inequalities (QVI), EMP, 1191
quoted
  names of sets, 542
text, 537
quotes, 542
RAMSEY, example from GAMSlib, 613
randBinomial, function, 588
randLinear, function, 588
random parameter
  DECIS, 1667
random variables
  DE, 1654
randTriangle, function, 588
range
  suffixes, equation, 632
  suffixes, variable, 618
range of numbers, 538
range statistics, GAMS output, 757
rayleigh, distribution, 1136
readyCollect, function, 596
ref, reference type, 750
refactorization
  CPLEX, 1588
reference file viewer, GAMS Studio, 2705
reformulation
  disjunctive program as MCP, 1214
  bilevel as MPCC, JAMS, 1893
  chance constraints, DE, 1659
  chance constraints, EMP, 1243
  CVaR, DE, 1660
  CVaR, EMP, 1239
  ENLP as MCP, JAMS, 1891
  ENLP as NLP, JAMS, 1891
  ENLP as NLP, theory, JAMS, 1903
  ENLP as VI, JAMS, 1901
  JAMS, 1888
  NLP as MCP, JAMS, 1891
  techniques, DE, 1657
  VaR, DE, 1659
  VaR, EMP, 1236
  VI as complementarity problem, JAMS, 1895
regularization
  DECIS, 1669
rel_eq, function, 592
rel_ge, function, 592
rel_gt, function, 592
rel_le, function, 592
rel_lt, function, 592
rel_ne, function, 593
repeat
  example, 690
  statement, 689
  syntax, 689
repeat, keyword, 534
report summary, GAMS output, 764
reporting format
  for compilation errors, 767
reserved non-alphanumeric symbols, 535
  **, 535
  ++, 535
  -, 535
  ->, 535
  .., 535
  =b=, 535
  =c=, 535
  =e=, 535
  =g=, 535
  =l=, 535
  =n=, 535
  =x=, 535
  <=>, 535
reserved words, see also keywords, 532
resource interrupt, solver status, 761
return codes, error codes, 1161
risk measures
  DE, 1656
  EMP, 1230
RLT
  ANTIGONE, 1412
RMINLP, model type, 637
RMIP, model type, 637
RMQCP, model type, 637
RMPEC, model type, 637
round, function, 588
rPower, function, 589
rules
  constructing tables, 606
  formatting tables, 606
sameas, 660
sameas, keyword, 534
sameAs, predefined symbol, 599
sampling
  DE, 1655
  EMP, 1223
  GAMS Stochastic Library, 1138
  LINDO Sampling Library, 1139
sand, keyword, 534
sand, operator, 582
SBB
  branch-and-bound algorithm, 2239
  pseudo costs, 2239
  SBB and DICOPT comparison, 2248
SBB and DICOPT comparison
  SBB, 2248
SBB, solver, 2239
scalar, 602
  example, 602
  statement, 602
  syntax, 602
scalar model
  CONVERT, 1555
  JAMS, 1888
scalar, keyword, 534
scalars, keyword, 534
scale
  option, 735
  suffixes, equation, 631
  suffixes, variable, 617

scaling
  models, 734
  of a variable, 735
  of an equation, 736
  of derivate, 737

scenario analysis, 653
scenario tree
  DE, 1654
scenario, keyword, 535
scenarios
  DE, 1654
  EMP, 1221
  scenred, tool, 2949
  scenred2, tool, 2956
SCIP
  emphasis settings, 2250
  indicators, 2250
  SCIP interactive shell, 2250
  solution pool, 2252
  tracing the solving process, 2252
SCIP interactive shell
  SCIP, 2250
SCIP, solver, 2249
SELKIE, solver, 2441

semi-continuous variables
  Definition, 730
  Example, 730

semi-integer variables
  definition, 731
  example, 731

semicolon, 537
semicont, keyword, 534
semiint, keyword, 534

sensitivity analysis
  CPLEX, 1591
  GUROBI, 1743

Sequential Linear Programming (SLP)
  CONOPT, 1493

Sequential Quadratic Programming (SQP)
  CONOPT, 1493

set, 541
  arithmetic, 562
  associated text, 542
  attributes, 552
  conditional, 665
  declaration for multiple sets, 543
  definition, 541
  domain defining symbol declarations, 556
  dynamic, 558
  elements, 541
  implicit set definition, 556
  matching operator, 547
  multi-dimensional, 546
  multi-dimensional many to many, 546
  multi-dimensional one-to-one mapping, 546
  names, 541
  sequences as set elements, 542
  simple, 541
  singleton, 549
  static, 559
  syntax, 541
  universal, 550
  using previously defined sets in set definitions, 543

set attributes, 552
set dictionary
  GUSS, 1804
set operations
  complement, 562
  difference, 562
  intersection, 562
  union, 562
set, keyword, 534
setMode, function, 1146
sets, keyword, 534
sets, see set, 541

SHALE, example from GAMSlib, 542
shellexecute, tool, 2968
SHOT, solver, 2444
sifting algorithm
  CPLEX, 1570
  sigmoid, function, 589
  sign, function, 589
  signPower, function, 589
  simple assignment, 578
  simplex algorithm
  CPLEX, 1570
  simplex optimizer
  MOSEK, 2086
sin, function, 589
sine, function, 1146
singleton, keyword, 534
sinh, function, 589
sj
  suffixes, put-file, 1000
slack
  suffixes, equation, 632
  suffixes, variable, 618
slacklo
  suffixes, equation, 632
  suffixes, variable, 618
slackup
  suffixes, equation, 632
  suffixes, variable, 618
slash, delimiter, 538, 991
sleep, function, 596
slexp, function, 589
sllog10, function, 589
slrec, function, 589
smax, keyword, 534
smax, operator, 582
smin, keyword, 534
<table>
<thead>
<tr>
<th>Index Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>smin, operator</td>
<td>582</td>
</tr>
<tr>
<td>SNOPT, solver</td>
<td>2458</td>
</tr>
<tr>
<td>soft constraints, EMP</td>
<td>1186</td>
</tr>
<tr>
<td>solution listing, GAMS output</td>
<td>762</td>
</tr>
<tr>
<td>solution pool</td>
<td></td>
</tr>
<tr>
<td>BARON, 1419</td>
<td></td>
</tr>
<tr>
<td>CPLEX, 1571</td>
<td></td>
</tr>
<tr>
<td>SCIP, 2252</td>
<td></td>
</tr>
<tr>
<td>solution pool, CPLEX, 1571</td>
<td></td>
</tr>
<tr>
<td>accessing, 1573</td>
<td></td>
</tr>
<tr>
<td>enumerating all solutions, 1572</td>
<td></td>
</tr>
<tr>
<td>filtering, 1573</td>
<td></td>
</tr>
<tr>
<td>populating, 1571</td>
<td></td>
</tr>
<tr>
<td>solution report, GAMS output</td>
<td>758</td>
</tr>
<tr>
<td>solve</td>
<td></td>
</tr>
<tr>
<td>error messages, 767</td>
<td></td>
</tr>
<tr>
<td>errors, 768</td>
<td></td>
</tr>
<tr>
<td>solve processing skipped, solver status</td>
<td>761</td>
</tr>
<tr>
<td>solve statement, 649</td>
<td></td>
</tr>
<tr>
<td>actions triggered by, 650</td>
<td></td>
</tr>
<tr>
<td>loop, 652</td>
<td></td>
</tr>
<tr>
<td>requirements, 650</td>
<td></td>
</tr>
<tr>
<td>several in a program, 651</td>
<td></td>
</tr>
<tr>
<td>several models, 651</td>
<td></td>
</tr>
<tr>
<td>syntax, 649</td>
<td></td>
</tr>
<tr>
<td>solve summary, GAMS output</td>
<td>758</td>
</tr>
<tr>
<td>evaluation errors, 759</td>
<td></td>
</tr>
<tr>
<td>iteration count, 759</td>
<td></td>
</tr>
<tr>
<td>model status, 759</td>
<td></td>
</tr>
<tr>
<td>objective summary, 759</td>
<td></td>
</tr>
<tr>
<td>resource usage, 759</td>
<td></td>
</tr>
<tr>
<td>solver status, 761</td>
<td></td>
</tr>
<tr>
<td>solve trace</td>
<td></td>
</tr>
<tr>
<td>Solver Usage, 1045</td>
<td></td>
</tr>
<tr>
<td>solve, keyword</td>
<td>534</td>
</tr>
<tr>
<td>solved singular, model status, 760</td>
<td></td>
</tr>
<tr>
<td>solved unique, model status, 760</td>
<td></td>
</tr>
<tr>
<td>solved, model status, 760</td>
<td></td>
</tr>
<tr>
<td>solver option editor, GAMS Studio, 2706</td>
<td></td>
</tr>
<tr>
<td>solver option file</td>
<td></td>
</tr>
<tr>
<td>solver option editor, 2706</td>
<td></td>
</tr>
<tr>
<td>Solver Usage, 1035</td>
<td></td>
</tr>
<tr>
<td>solver status, 761</td>
<td></td>
</tr>
<tr>
<td>capability problems, 761</td>
<td></td>
</tr>
<tr>
<td>error internal solver failure, 761</td>
<td></td>
</tr>
<tr>
<td>error setup failure, 761</td>
<td></td>
</tr>
<tr>
<td>error solver failure, 761</td>
<td></td>
</tr>
<tr>
<td>error system failure, 761</td>
<td></td>
</tr>
<tr>
<td>evaluation interrupt, 761</td>
<td></td>
</tr>
<tr>
<td>iteration interrupt, 761</td>
<td></td>
</tr>
<tr>
<td>licensing problems, 761</td>
<td></td>
</tr>
<tr>
<td>normal completion, 761</td>
<td></td>
</tr>
<tr>
<td>resource interrupt, 761</td>
<td></td>
</tr>
<tr>
<td>solve processing skipped, 761</td>
<td></td>
</tr>
<tr>
<td>terminated by solver, 761</td>
<td></td>
</tr>
<tr>
<td>user interrupt, 761</td>
<td></td>
</tr>
<tr>
<td>solver statuses</td>
<td></td>
</tr>
<tr>
<td>BARON, 1417</td>
<td></td>
</tr>
</tbody>
</table>

**Solver Usage**

- BCH, examples, 1050
- Branch-and-Cut-and-Heuristic facility (BCH), 1046
- GAMS dot options for solvers, 1036
- GAMS options for solvers, 1034
- options, global, 1034
- options, model-specific, 1034
- options, solver, 1034
- solve trace, 1045
- solver option file, 1035
- solverCapabilities, predefined symbol, 599
- solvers, 1383
- AlphaECP, 1387
- ANTIGONE, 1402
- BARON, 1414
- CBC, 1429
- CONOPT, 1460
- CONOPT4, 1521
- CONVERT, 1555
- CPLEX, 1569
- DE, 1653
- DECIS, 1665
- DICOPT, 1693
- EXAMINER, 1718
- GAMSCHK, 1722
- GUROBI, 1743
- GUSS, 1804
- HIGHS, 1820
- IPOPT, 1826
- JAMS, 1888
- KESTREL, 1906
- KNITRO, 1909
- LINDO, 1962
- LINDOGLOBAL, 1962
- MILES, 2033
- MINOS, 2053
- MOSEK, 2086
- NLPEC, 2139
- OCTERACT, 2151
- ODHCPLEX, 2163
- OsiCplex, 2199
- OsiGurobi, 2199
- OsiMosek, 2199
- OsiXpress, 2199
- PATH, 2205
- PATHNLP, 2202
- SBB, 2239
- SCIP, 2249
- SELKIE, 2441
- SHOT, 2444
- SNOPT, 2458
- SOPLEX, 2493
- XA, 2499
- XPRESS, 2508
- solvers, predefined symbol, 599
- SOPLEX, solver, 2493
- sor, keyword, 534
sor, operator, 582
sortedUels, predefined symbol, 599
sos1, keyword, 534
sos2, keyword, 534
special ordered sets
  type 1 - definition, 728
  type 1 - example, 729
  type 2 - definition, 729
sqexp, function, 589
sql2gms, tool, 2973
sqllog10, function, 589
sqr, function, 589
sqrec, function, 590
sqrt, function, 590
stage
  suffixes, equation, 631
  suffixes, variable, 617
stages
  DE, 1654
  EMP, 1218
Standard Locations, 310
starting solution for MIPs
  CPLEX, 1590
  GUROBI, 1744
statements, list of, 527
static set, 559
Steepest Edge Algorithm
  CONOPT, 1493
stochastic linear program
  DE, 1653
  DECIS, 1666
  EMP, 1218
stochastic linear program, two-stage
  DE, 1657
  DECIS, 1666
stochastic parameters, dependent
  DECIS, 1675
stochastic parameters, independent
  DECIS, 1672
Stochastic Programming
  LINDO, 2032
Stochastic programming
  DECIS, 1666
stochastic programming
  DE, 1653
  EMP, 1217
Stochastic Programming, EMP
  DECIS, 1666
stochastic programming, EMP, 1217
  DE, 1653
studentT distribution
  GAMS Stochastic Library, 1136
  LINDO Sampling Library, 1140
Studio, GAMS Tool, 2691
subsets, 580
subsolver
  JAMS, 1888
subsolvers not compatible with GUSS
  GUSS, 1804
suffix
  function, 596
  put-file, 995
  variable, 616
suffixes, equation, 631
  .infeas, 632
  .l, 631
  .lo, 631
  .m, 631
  .range, 632
  .scale, 631
  .slack, 632
  .slacklo, 632
  .slackup, 632
  .stage, 631
  .up, 631
suffixes, function, 596
  .grad, 597
  .gradH, 597
  .gradL, 597
  .gradn, 597
  .hess, 597
  .hessn, 597
  .high, 597
  .low, 597
  .value, 597
suffixes, identifier (put context), 1009
  .te, 1010
  .tl, 1009
  .tn, 1010
  .ts, 1009
suffixes, put-file, 995
  .ap, 996
  .bm, 996
  .case, 996
  .cc, 996
  .cr, 996
  .errors, 996
  .hdcc, 996
  .hdr, 997
  .hdll, 997
  .lcase, 997
  .lj, 997
  .ll, 997
  .lm, 997
  .lw, 997
  .nd, 997
  .nj, 998
  .nr, 998
  .nw, 998
  .nz, 998
  .pc, 999
  .pdir, 998
  .ps, 1000
  .pw, 1000
  .sj, 1000
INDEX

 swath, 1000
 tf, 1001
tj, 1001
tlcc, 1001
tlcr, 1001
tll, 1001
tm, 1002
tw, 1002
 ws, 1002

 suffixes, string type, 590
 .te, 591
 .tl, 591
 .ts, 591

 suffixes, variable, 616
 .fx, 616
 .infeas, 618
 .l, 617
 .lo, 616
 .m, 617
 .prior, 617
 .range, 618
 .scale, 617
 .slack, 618
 .slacklo, 618
 .slackup, 618
 .stage, 617
 .up, 616

 sum, keyword, 534
 sum, operator, 582

 superbasic variable, 763
 sw
 suffixes, put-file, 1000
 symbol listing map, GAMS output, 751
 symbol mapping information
 GUSS, 1804
 symbol reference map, GAMS output, 749
 acrnm, 750
 assigned, 750
 control, 750
 declared, 750
 defined, 750
 equ, 750
 file, 750
 impl-asn, 750
 model, 750
 param, 750
 ref, 750
 set, 750
 var, 750

 system
 attributes, 709
 system, keyword, 534

table, 605
 condensing, 608
 continued, 607
 example, 606
 long row labels, 609
 more than two dimensions, 607

 rules for constructing, 606
 rules for formatting, 606
 statement, 605
 syntax, 605

table, keyword, 534
tan, function, 590
tanh, function, 590
t e
 suffixes, identifier (put context), 1010
 suffixes, string type, 591

terminated by solver, solver status, 761
 termination message, NaN (Not A Number)
 CONOPT, 1500
 termination message, overflow
 CONOPT, 1500
 termination message, stalling
 CONOPT, 1500
 termination messages
 BARON, 1417
 CONOPT, 1463

 TeX
 MODEL2TEX, 2941
 tf
 suffixes, put-file, 1001
 then, keyword, 534
 timeClose, function, 596
timeComp, function, 596
timeElapsed, function, 596
timeExec, function, 596
timeStart, function, 596
 tj
 suffixes, put-file, 1001
 tl
 suffixes, put-file, 1001

 tlcc
 suffixes, put-file, 1001
 tllcr
 suffixes, put-file, 1001
 tll
 suffixes, put-file, 1001
 tm
 suffixes, put-file, 1002
 tn
 suffixes, identifier (put context), 1010

tools, 2654
 ask, 2654
 chk4upd, 2654
 cholesky, 2654
csdp, 2654
csv2gdx, 2654
eigenvalue, 2654
eigenvector, 2654
endecrypt, 2654
findthisgams, 2654
GAMS IDE, 2654
GAMS Studio, 2654
gdx2access, 2654


3297
Tutorials
.NET API Tutorial, 3164
C++ API Tutorial, 3180
GAMS Tutorials, 318
Java API Tutorial, 3198
Matlab API Tutorial, 3233
Python API Tutorial, 3218
tw
suffixes, put-file, 1002
type
of discrete variables, 727
unbounded, 763
unbounded - no solution, model status, 760
unbounded, model status, 759
uncertainty
DECIS, 1667
undf, extended range value, 598
undf, keyword, 534
uniform distribution
GAMS Stochastic Library, 1136
LINDO Sampling Library, 1140
uniform, function, 590
uniformInt, distribution, 1137
uniformInt, function, 590
union, of sets, 562
unique element listing map, GAMS output, 751
universal set, 550
universe, 550
universe problem
DECIS, 1668
UNIX Installation Notes, 305
until, keyword, 534
up
suffixes, equation, 631
suffixes, variable, 616
updating parts of a model iteratively
GUSS, 1804
user interrupt, solver status, 761
using, 649
using, keyword, 535
value
suffixes, function, 597
variable
binary, 613
discrete, 613
free, 613
integer, 613
negative, 613
positive, 613
statement, 613
statements, 613
styles for declaration, 615
suffix, 616
syntax of declaration, 612
types, 613
variable attributes, see suffixes, variable, 616
variable bounds, 618
activity level, 619
BARON, 1415
fixing, 619
lower, 619
variable, keyword, 534
variables, keyword, 534
variational inequalities (VI), EMP, 1188
variational inequalities (VIs)
   JAMS, 1895
vcPower, function, 590
weibull distribution
   GAMS Stochastic Library, 1136
   LINDO Sampling Library, 1140
while
   example, 686
   statement, 686
   syntax, 686
while, keyword, 534
Windows Installation Notes, 307
ws
   suffixes, put-file, 1002
XA, solver, 2499
XLS2GMS
   $CALL command, 3027
   command files, 3027
   example, simple, 3017
   multiple area ranges, 3028
   post processing, 3028
   sets, importing, 3018
   tables, importing, 3019
   tables, multidimensional, importing, 3021
   use, command line, 3025
   use, interactive, 3022
xls2gms, tool, 3016
xlsdump, tool, 3032
xlstalk, tool, 3033
xor, keyword, 534
XPRESS
   XPRESS-Link, 2652
XPRESS, solver, 2508
XPRESS-Link
   XPRESS, 2652
yes, keyword, 534
Bibliography


